NMS OAM System User's Manual

P/N 9000-6819-21

NMS COMMUNICATIONS

100 Crossing Boulevard
Framingham, MA 01702-5406 USA
www.nmscommunications.com

P/N 9000-6819-21

Revision history

Revision	Release date	Notes
1.0	February, 2000	CYF, for Natural Access 2000-1 Beta
1.1	June, 2000	CYF, for PSF 4.0
1.2	September, 2000	CYF, for Natural Access 2000-1
1.3	February, 2001	CYF, for 2000-2 maintenance release
1.4	April, 2001	CYF, for 2001-1
1.5	June, 2001	CYF
1.6	August, 2001	CYF, for NACD 2001-1
1.7	November, 2001	CYF, for NACD 2002-1 Beta
1.8	May, 2002	MVH, for NACD 2002-1
1.9	April, 2003	SRR, for NACD 2003-1
2.0	November, 2003	MVH, for Natural Access 2004-1 Beta
2.1	April, 2004	MCM, for Natural Access 2004-1
Last modified: April 6, 2004		

Refer to the NMS web site (www.nmscommunications.com) for product updates and for information about support policies, warranty information, and service offerings.

Table Of Contents

1 Introduction

The *NMS OAM System User's Manual* describes how to set up a chassis containing NMS boards, and how to use NMS OAM software to configure, start, and monitor the boards.

This document is written for developers and system administrators.

2 Overview of NMS OAM

NMS OAM components

NMS Communications operations, administration, and maintenance (OAM) software is the component of the Natural Access development environment that enables you to administer and maintain NMS resources in a system. NMS OAM can manage hardware components such as NMS boards, or software components such as the NMS Hot Swap and H.100/H.110 clock management processes.

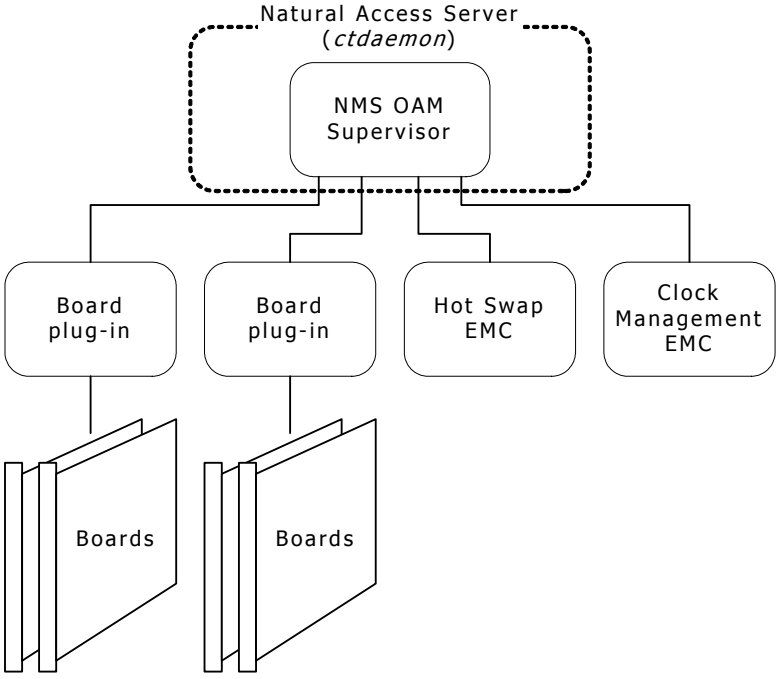
This topic presents:

- NMS OAM architecture
- Performing OAM tasks

NMS OAM architecture

NMS OAM software includes the following components:

- NMS OAM Supervisor (a part of the Natural Access Server *ctdaemon*)
- Board plug-ins
- Extended management components (EMCs)



NMS OAM components

NMS OAM Supervisor

The NMS OAM Supervisor provides the main NMS OAM logic. It performs the following tasks:

- Loads all board plug-ins and extended management components (EMC) when it starts up.
- Coordinates the activities of the managed components in the system.
- Manages a database containing configuration information for the components in the system.

The NMS OAM Supervisor is an integral part of the Natural Access Server (*ctdaemon*). To use the NMS OAM software, Natural Access must be installed on your system and *ctdaemon* must be running. To learn how to start *ctdaemon*, refer to *Starting the Natural Access Server* on page 33.

Board plug-ins

NMS OAM communicates with boards through software extensions called board plug-ins, one for each board family. The NMS OAM board plug-ins support the AG, CG, CX, and QX PCI and CompactPCI board models. The board plug-ins do not support TX boards.

When the Supervisor starts up, it loads all plug-ins that it finds. The Supervisor looks for these modules in the *\nms\bin* directory (*/opt/nms/lib* under UNIX). Plug-in files have the extension *.bpi*.

Extended management components (EMCs)

EMCs are software modules that add functionality to NMS OAM. NMS OAM provides the following EMCs:

- The Hot Swap EMC allows you to insert and extract Hot Swap-compatible CompactPCI boards without powering down the system. Hot Swap improves system availability by reducing down-time due to routine configuration changes and board replacements.
- The Clock Management EMC manages H.100 and H.110 bus clock configurations.

When the Supervisor starts up, it loads all EMCs that it finds. The Supervisor looks for these modules in the *\nms\bin* directory (*/opt/nms/lib* under UNIX). EMC files have the extension *.emc*.

Performing OAM tasks

You can perform the following tasks using NMS OAM:

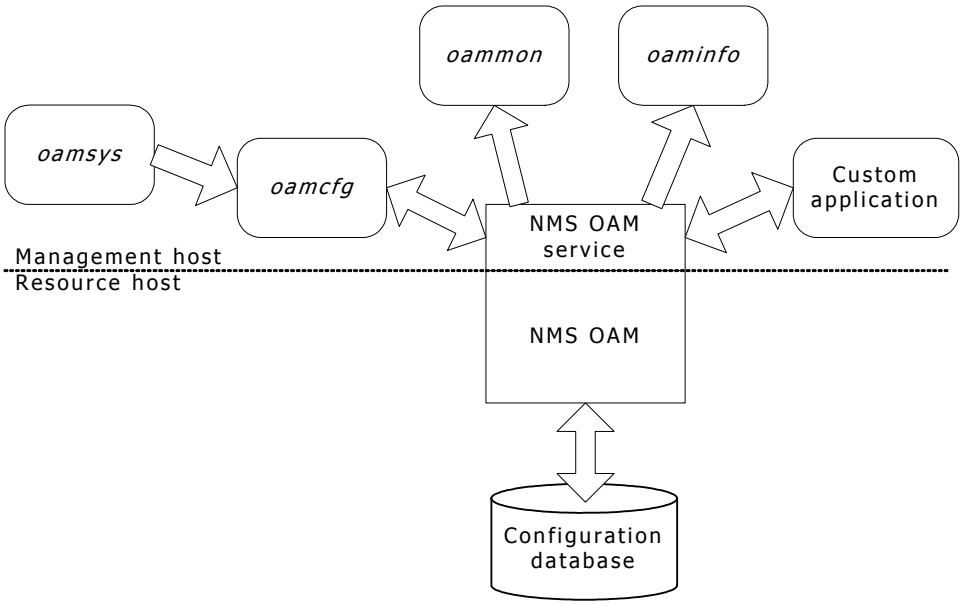
- Create, delete, and query component configurations
- Start, stop, and test components
- Receive notifications from components

Use the following NMS OAM components to perform OAM tasks:

- The *oamsys*, *oamcfg*, *oammon*, and *oaminfo* utilities.
- The NMS OAM service.

In a multiple-host NMS OAM configuration, these utilities reside on the management host and communicate with the resource hosts over the IP network.

The following illustration shows the relationships between these NMS OAM components:



NMS OAM utilities and NMS OAM service

To use any NMS OAM utility on any host, the Natural Access Server (*ctdaemon*) must be running on the target host and have the NMS OAM Supervisor started within it. For more information, refer to *Starting the Natural Access Server* on page 33.

oamsys

To configure and start managed components on a host, use the *oamsys* utility. This utility creates records for components in the NMS OAM database on a resource host based on system configuration files you supply. It then attempts to start all boards in the database.

Configuration parameter values for each managed component are listed in the system configuration file. If the component is a board, this information includes the board's ID information.

oamsys completely renews the database each time it runs and restarts all boards. Any parameters not listed in the configuration file are reset to their default settings. Thus *oamsys* makes it easy to track the configuration of an entire host.

Note: If you are migrating to NMS OAM, utilities are available to assist in the transition from *agmon* to NMS OAM. For details, see *Summary of changes* on page 119.

To perform its tasks, the *oamsys* utility makes multiple calls to the *oamcfg* utility.

Use *oamsys* to set up both local and remote configurations. For more information, see *Using oamsys* on page 53.

oamcfg

oamcfg provides access to individual NMS OAM configuration functions. Using this utility, you can:

- Create or delete boards in the database
- Specify settings for a component's parameters, either individually or collectively, using keyword files
- Start or stop one or more boards
- Test boards (if supported)
- Detect boards in a system
- Display basic ID information for boards
- Import or export the contents of the NMS OAM database

oamcfg can perform one or more operations for a single component. Alternatively, the utility can perform operations on all board components in the database with one call. All *oamcfg* operations can be performed on both local and remote resources.

Use *oamcfg* to update components. *oamcfg* can be cumbersome if you use it to update many components in a complex system. Use *oamsys* for this purpose.

For more information on *oamcfg,* refer to *oamcfg overview* on page 57.

oammon

The *oammon* utility enables you to access NMS OAM monitoring functions. Using *oammon*, you can:

- Monitor board errors and other messages
- Capture messages in a flat file
- Send a test alert notification message to all NMS OAM client applications

oammon can monitor both local and remote resources. For more information, refer to *Using oammon* on page 54.

oaminfo

The *oaminfo* utility enables you to access keywords from the command line. *oaminfo* can display all keywords for a component, or specific keywords and values. It can also search for text in keyword names and set keyword values.

oaminfo can perform its functions on both local and remote hosts. For more information about *oaminfo*, refer to the *NMS OAM Service Developer's Reference Manual*.

NMS OAM service

You can program access to NMS OAM functionality using the NMS OAM service. NMS OAM is implemented as a service under the Natural Access development environment. Natural Access provides standard programming interfaces for hardware-independent functions. Under Natural Access, logically related functions (NMS OAM operations, for example) are divided into groups called services, which have similar APIs.

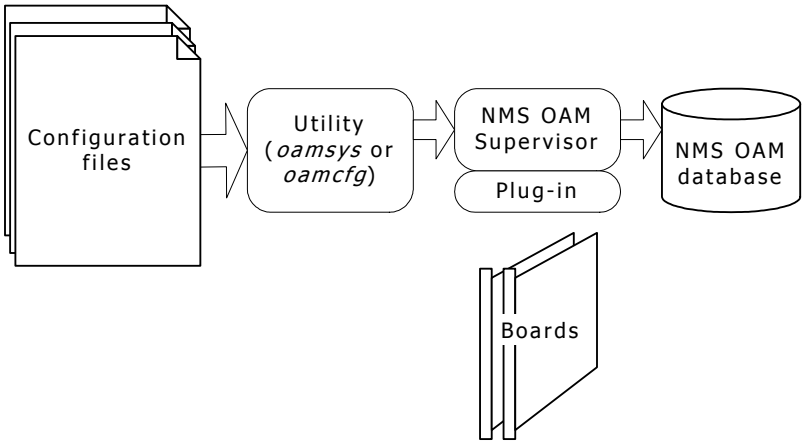
NMS OAM utilities make calls to the NMS OAM service to perform their operations.

For detailed information about using the NMS OAM service, refer to the *NMS OAM Service Developer's Reference Manual*.

NMS OAM database

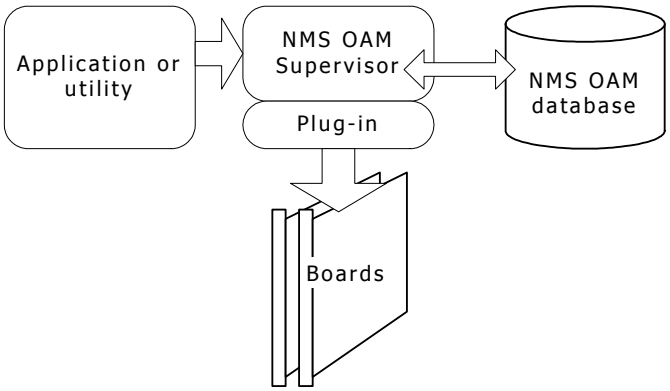
NMS OAM maintains a database containing configuration information for each component under its control. This information consists of parameters and values. Each parameter and value is expressed as a keyword name and value pair (for example, AutoStart = YES). You can use NMS OAM to retrieve and modify configuration parameters.

When you set up your system, you build text files describing the components. You enter this information into the database using NMS OAM utilities such as *oamsys* and *oamcfg*:



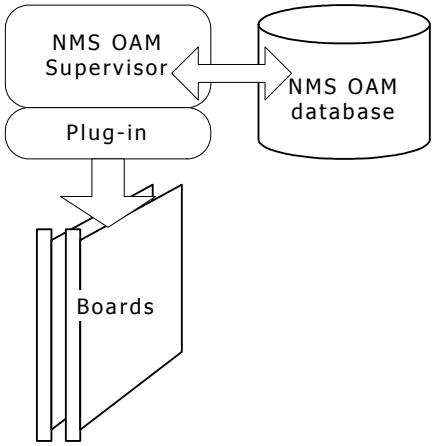
Configuring the NMS OAM database

With the utilities or an application based on the NMS OAM service, you use NMS OAM to configure, start, and manage components in the system based on the NMS OAM database:



Using configuration information

You can also set up NMS OAM to configure and start boards automatically whenever it starts up, without any intervention by a utility or application. See the following illustration:



Auto-starting boards

Board identification methods

Within NMS OAM, each board is referenced using the following identifiers:

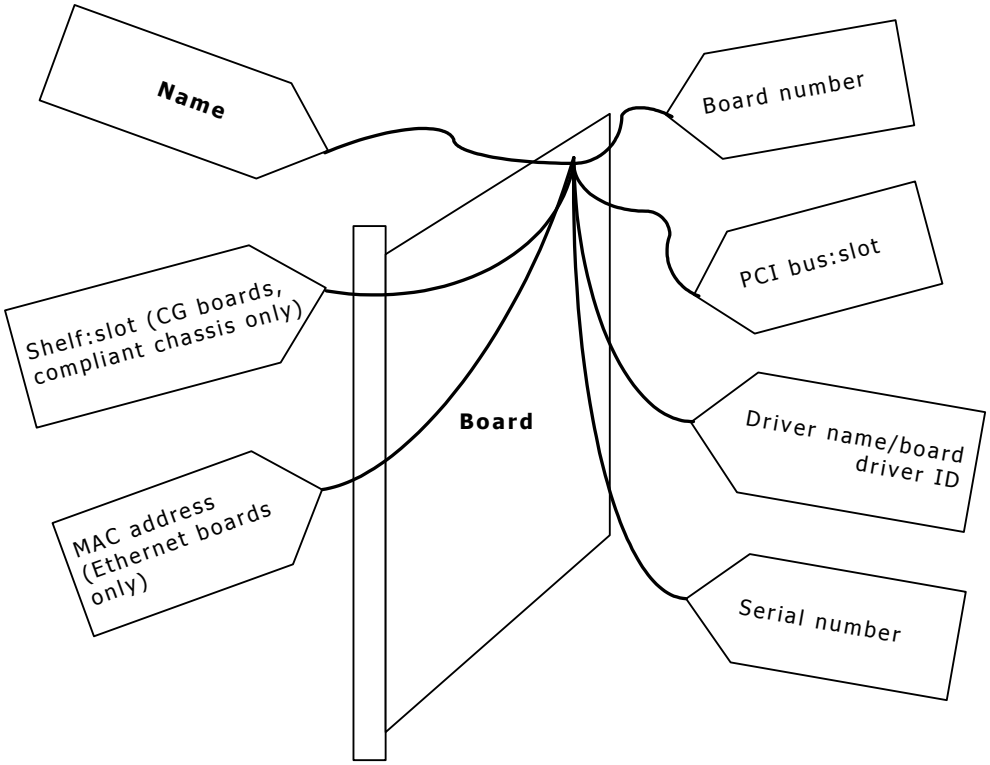
- A unique name.
- A board number. Each board in a host has a unique board number.
- A unique PCI bus and slot in which the board is located.

The following secondary ID information is also available:

- A driver name and driver board ID combination. The driver name is unique among all driver names on the host. The driver board ID is unique among all boards accessed by a given driver. However, two boards accessed by different drivers can have the same driver board ID. The driver name and driver board ID together make up an ID for the board that is unique within the host.
- A serial number if supported. This number is assigned at the factory, and is not present for all boards.
- CompactPCI CG and CX boards only. A unique shelf and slot in which the board is located. The shelf refers to the backplane or portion of the backplane in which the board is installed. Slot refers to the physical slot within the chassis where the board is located.

  Note: Implementation of shelf and slot differs depending upon the chassis manufacturer and specific hardware settings. Shelf and slot information is available only with boards installed in CompactPCI chassis with a bus that complies with PICMG 2.1. For more information, refer to the chassis documentation.

- Boards with Ethernet capability only. Unique MAC addresses, one for each Ethernet controller on each board.



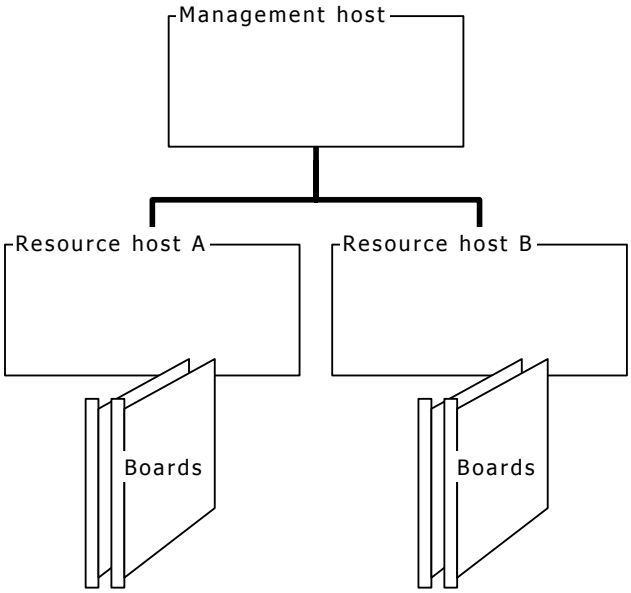
Board identification options

In a multiple-host NMS OAM configuration, it is possible to have boards in different hosts with the same board name and board number. However, to avoid confusion, give each board a name and number that is unique to the entire host.

Multiple-host NMS OAM

If your resources are distributed over several systems (multiple CPUs, chassis, or both) that are linked over an IP network, you can set up a multiple-host configuration of NMS OAM. This configuration enables an application running on one system to access and manage resources on other systems.

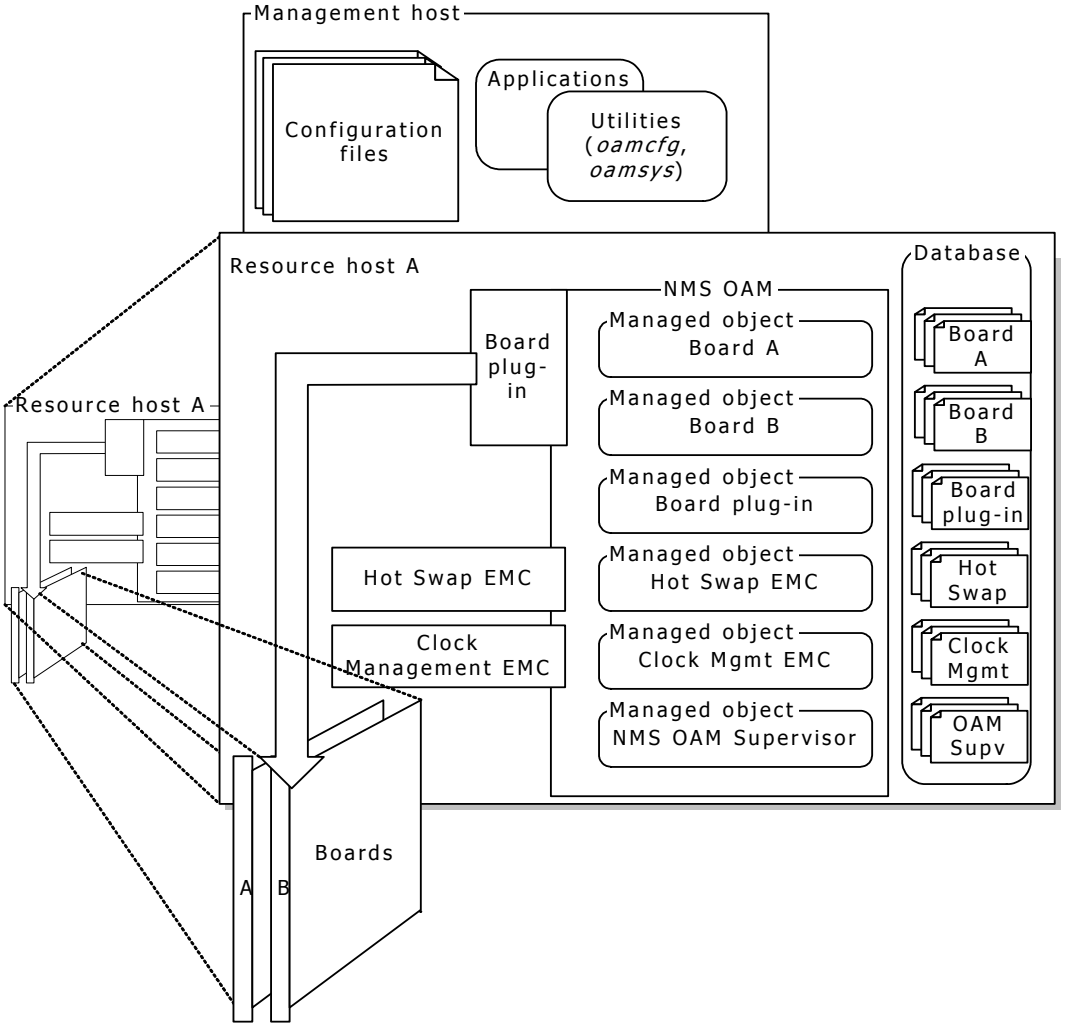
A multiple-host NMS OAM configuration consists of several hosts (CPUs). One of these hosts, the management host, configures and manages the resources (such as boards) on other hosts that are called resource hosts. Refer to the following illustration:



Management host and resource hosts

Each resource host runs an instance of the Natural Access Server (*ctdaemon*), including NMS OAM. Each resource host also has its own NMS OAM database containing data for the components on that host only.

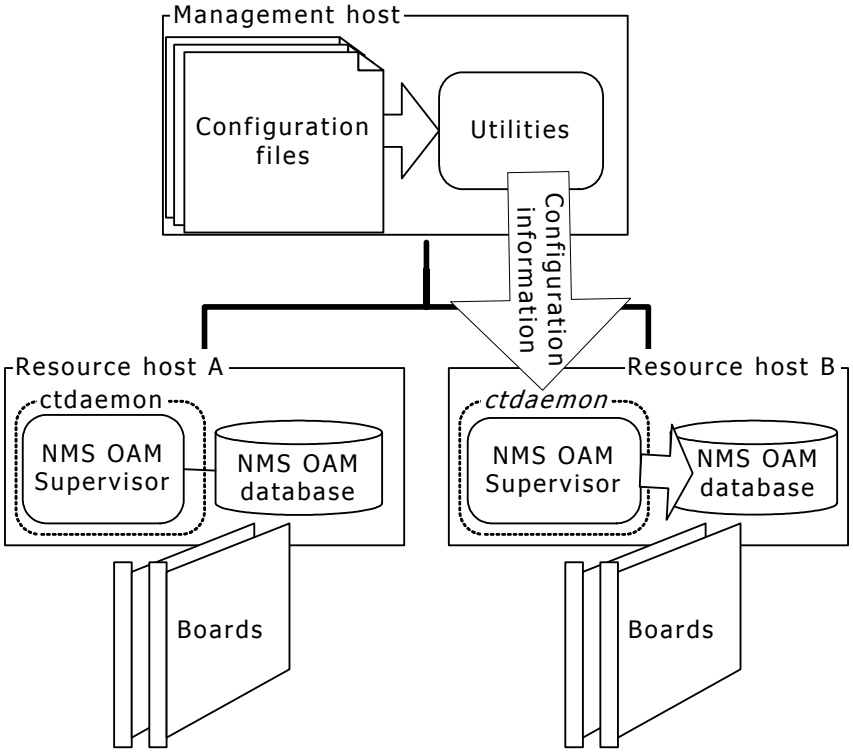
Management applications and utilities, such as the NMS OAM utilities, reside on the management host. The configuration text files also reside on the management host.



NMS OAM host components

If a management host includes resources that require management, the management host can also serve as a resource host. In this case, the management host also must run an instance of the Natural Access Server (*ctdaemon*), including NMS OAM. The database on this host contains information about the local resources only, even if the management host is also managing other resource hosts.

The utilities on the management host are used to configure the database on each resource host, one database at a time. Refer to the following illustration:



Configuring the NMS OAM database on a resource host

Applications on the management host can direct NMS OAM on a resource host to configure, start, and manage its resources based on the NMS OAM database.

Overview of setting up NMS OAM

Administrators can use NMS OAM to set up hosts in single host configurations or in multiple-host configurations.

Single-host configurations

If your configuration consists of only one host, follow these steps to set up NMS OAM:

Step	Description	Documented in...
1	Ensure that your chassis is set up properly for Hot Swap boards. (Required only if you are using Hot Swap.)	*Configuring Hot Swap* on page 21
2	Install NMS OAM software.	Natural Access installation booklet
3	Start the Natural Access Server (*ctdaemon*), if it is not already running. Also start Hot Swap.	*Starting the Natural Access Server* on page 33 *Starting Hot Swap* on page 31
4	Create a system configuration file describing your system. In this file, give each board a unique name and board number.	*Creating a system configuration file* on page 43
5	If your system contains two or more boards connected through the H.100 or H.110 bus, configure clocking on the bus.	*CT bus clocking overview* on page 87
6	Use *oamsys* to create records for components in the NMS OAM database based upon the system configuration file, and to start all installed boards.	*Using oamsys* on page 53 *Board identification methods* on page 14

Multiple-host configurations

To set up a multiple-host NMS OAM configuration, follow these steps:

Step	Description	Documented in...
1	Determine which systems will serve as resource hosts, and which one will serve as the management host.	
2	CompactPCI only. Ensure that each host chassis is set up properly for Hot Swap boards. (Required only if you are using Hot Swap.)	*Configuring Hot Swap* on page 21
3	Install NMS OAM software on each host.	Natural Access installation booklet
4	Start the Natural Access Server (*ctdaemon*) on each host, if it is not already running. Also start the Hot Swap driver and the Hot Swap manager.	*Starting the Natural Access Server* on page 33 *Starting Hot Swap* on page 31
5	Create system configuration files, one for each resource host. In each file, give each board in the host a unique name and board number.	*Creating a system configuration file* on page 43
6	If a host contains two or more boards connected through the H.100 or H.110 bus, configure clocking on the bus.	*CT bus clocking overview* on page 87
7	Use *oamsys* to create records for components in the NMS OAM database on each host based upon the system configuration file, and to start all installed boards.	*Using oamsys* on page 53 *Board identification methods* on page 14

3 Setting up an NMS OAM host

Configuring Hot Swap

Hot Swap functionality is an integral part of NMS OAM. It is designed for use with CompactPCI Hot Swap-compliant boards. It is supported on Windows and UNIX systems.

This topic provides the following information:

- Hot Swap-compatible boards
- Hot Swap EMC
- Hot Swap platform requirements
- Configuring Hot Swap under Windows
- Configuring Hot Swap under UNIX

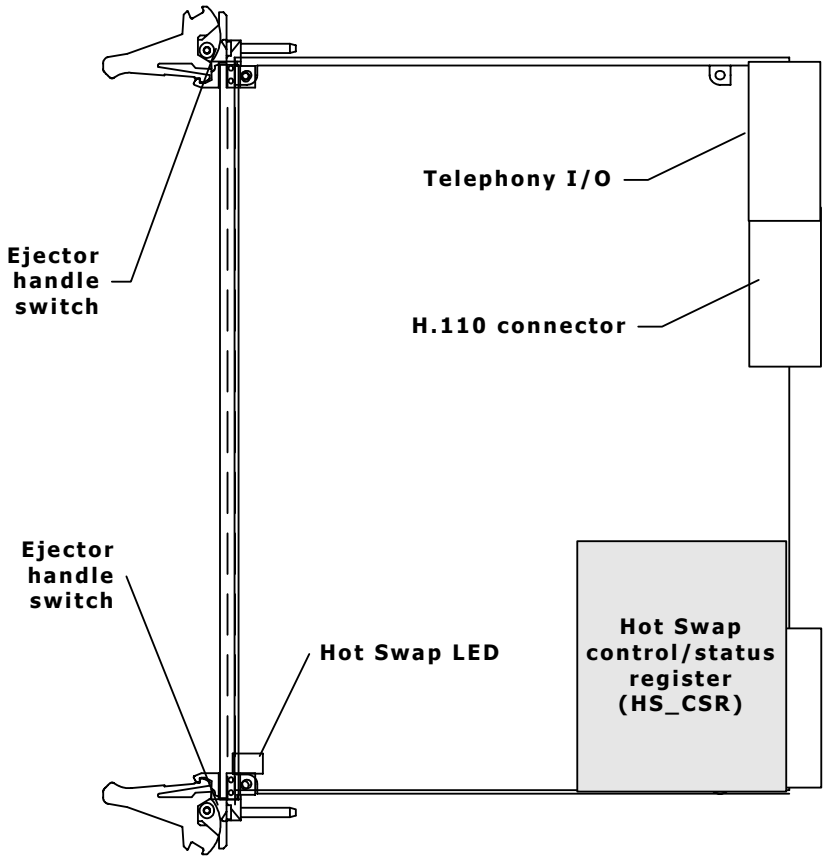
Hot Swap-compatible boards

A Hot Swap-compatible board includes a switch built into the ejector handle and a front panel Hot Swap LED. When you insert a board into the system, the switch signals that the board is fully seated with the handle closed and that the software connection can be initiated. When you remove a board, the switch signals that the board is being extracted and that the software disconnection can be initiated.

When lit, the Hot Swap LED indicates that the software disconnection is complete and extraction is permitted. You can open the handle the rest of the way and eject the board.

The PCI interface for NMS Hot Swap-compatible CompactPCI boards includes the Hot Swap control and status register (HS_CSR). The PCI interface is responsible for management of the ejector handle switches and the Hot Swap LED.

The following illustration shows the ejector handles and Hot Swap LED on an AG 4040C board:



AG 4040C board

Hot Swap EMC

Hot Swap is implemented as an extended management component (EMC). The Hot Swap EMC:

- Automatically stops a CompactPCI board prior to its physical removal from the chassis.
- Automatically starts a board when it is physically installed in the chassis (if supported). Board automatic starting is controlled by configuration keywords. For more information, refer to *Starting boards automatically* on page 51.
- Makes alerts and other messages related to Hot Swap available to client applications.

The Hot Swap EMC communicates with the Hot Swap manager and driver to perform Hot Swap operations. The Hot Swap manager and driver must be started in order for Hot Swap operations to work. To learn how to start these components, refer to *Starting Hot Swap* on page 31.

Note: Hot Swap is supported only by CompactPCI boards, but some CompactPCI boards do not support Hot Swap. Removing a board that does not support Hot Swap functionality while the system is running can cause serious damage to the board and to the system. To determine if a board model supports Hot Swap, refer to the documentation for the board.

Hot Swap platform requirements

Hot Swap development requires an Intel or SPARC CompactPCI-compliant platform that conforms to the following specifications:

- PICMG 2.0 Revision 2.1 CompactPCI
- PICMG 2.1 Revision 1.0 CompactPCI Hot Swap (Hot Swap platform)
- PCI BIOS Revision 2.1 (PCI BIOS services are used to manage interrupt assignments for hot-inserted boards.)
- PICMG 2.5 Revision 1.0 CompactPCI Computer Telephony (If the H.110 bus is not present, the CompactPCI board will not power up.)

Configuring Hot Swap under Windows

To configure Hot Swap under Windows:

1. Install the Hot Swap Kit during the Natural Access installation.

   To learn how to install the Hot Swap Kit, refer to the Natural Access installation booklet.

   Note: If you do not install the Hot Swap Kit, inserting or removing an NMS CompactPCI board while the system is running can damage the chassis or the board.
2. Shut down your system and turn the power off.
3. Remove all NMS CompactPCI boards from the system.
4. Power up the system and log in as a user with administrative privileges.
5. Run the HSK Wizard once for each system slot processor.

Running the HSK Wizard

The Hot Swap Kit Wizard performs two functions:

- Determines the logical slot number for each physical peripheral slot in a CompactPCI chassis. A slot path identifies each physical slot.
- Adjusts the resources assigned to the CompactPCI bridges in the system at boot time so that boards can be inserted and extracted while the system is running. The new resource allocations are stored in the registry and become effective whenever the system is booted.

  This resource adjustment is important because, by default, CompactPCI bridge memory windows are initialized to be just big enough for devices physically installed at boot time. Unless the wizard is run to set up a different configuration, only boards physically installed at boot time can be hot swapped in and out.

The wizard prompts you for the number of slots in the chassis. It then asks you to insert a board in each slot. When you insert the board, the wizard locates the board and maps the logical slot number to the physical slot. It then asks you to remove the board and insert it in another slot, and repeats the process.

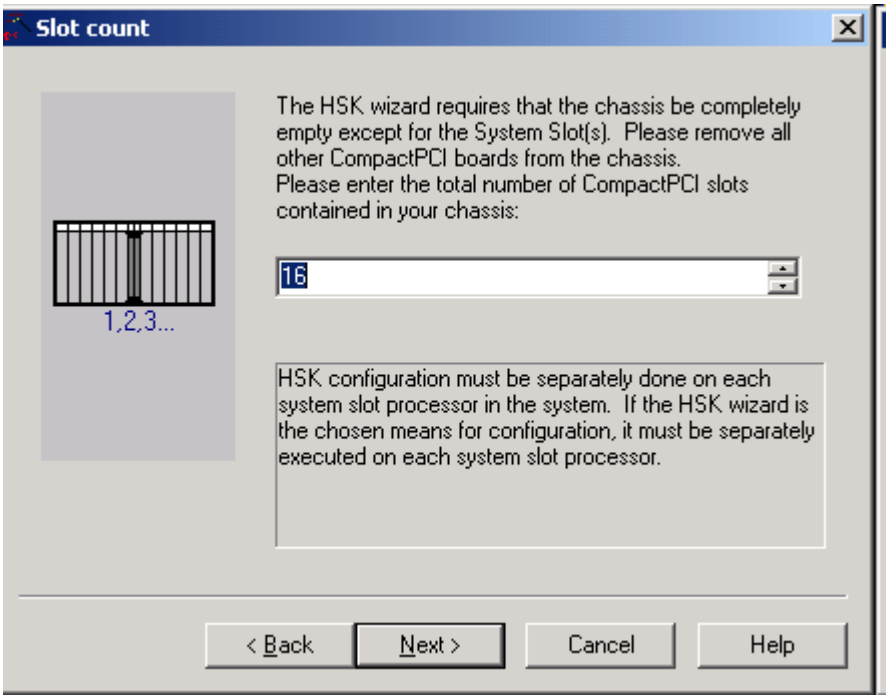
Configure each system slot processor in the system separately. If the CompactPCI bus is divided into multiple segments controlled by separate CPUs, you must run the wizard separately for each CPU. In this case, the slot numbers indicated on the chassis may not match the slot numbers indicated in the wizard.

You can use any NMS Hot Swap-compatible board in the configuration process.

To use the HSK Wizard:

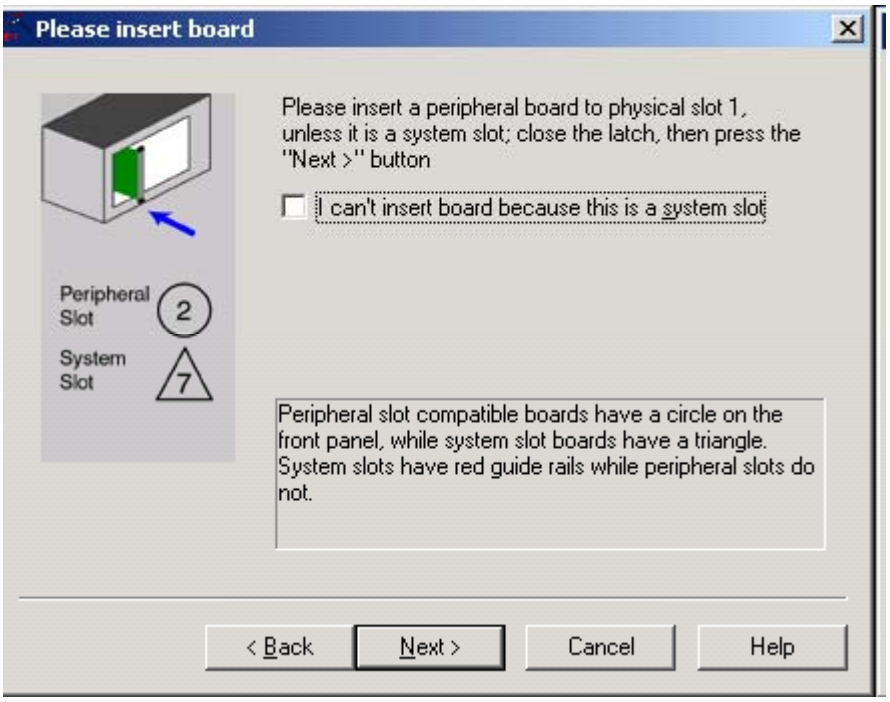
1. Launch the HSK Wizard by selecting **Start**->**Programs**->**Hot Swap Kit**->**HSK Wizard**.

   The Slot count dialog box appears:



2. Enter the total number of slots contained in the chassis including slots occupied by system components. Click **Next**.

   The Please insert board dialog box appears:



3. Insert a board into the specified slot, and close the ejector handles.

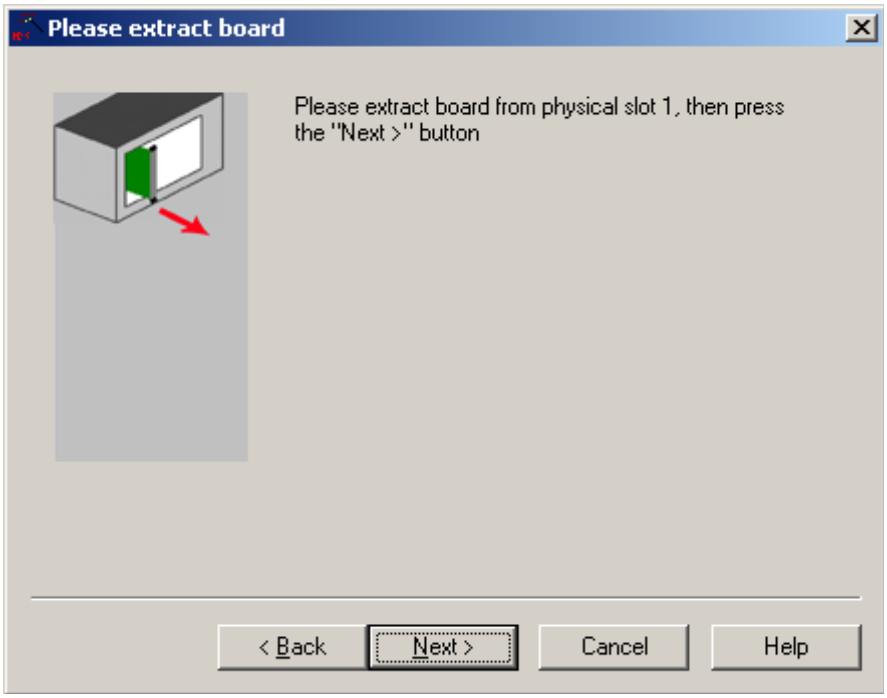
   If you cannot insert the board because the specified slot is a system slot, select **I can't insert board because this is a system slot**.

   If your CompactPCI bus is divided into multiple segments, the slot numbers indicated on the chassis may not match the slot numbers indicated in the wizard. In this case, you must determine which slots belong to the segment. When prompted for physical slot 1, use the left-most peripheral slot in the segment. When prompted for physical slot 2, use the slot to the right of this slot, and so on.

4. Click **Next**.

   If you selected **I can't insert board because this is a system slot**, the Please insert board dialog box appears prompting you to insert a board in another slot. Repeat step 3 for the new slot.

Otherwise, the wizard searches the CompactPCI bus and associates the specified slot number with the slot currently containing the board. The Please extract board dialog box appears:



5. Open the board's ejector handles.

   After a moment, the Hot Swap LED on the board lights.

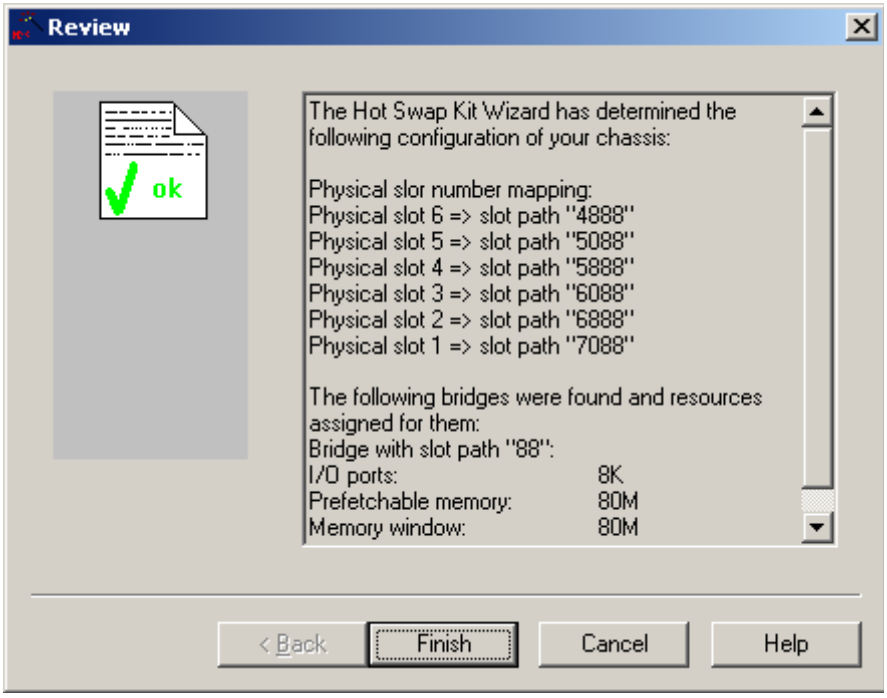
6. When the Hot Swap LED lights, extract the board.

   If you remove the board before the LED lights, you can damage the chassis or the board.

7. Click **Next**.

8. Repeat steps 3 through 7 for each slot in the system. Continue inserting and extracting boards as directed by the wizard.

After you have inserted and extracted a board from every slot, the Review dialog box displays the settings for each physical slot:



9. Click **Finish**.

   A dialog box displays a prompt to save your changes to the registry.

10. If you are satisfied with your changes, click **Yes**. Otherwise, click **No** and run the wizard again.

Windows Hot Swap Kit utilities

You can make modifications to the Hot Swap configuration and monitor Hot Swap activity using the following utilities installed with the Hot Swap Kit:

Utility	Description
Hot Swap Kit Configuration	• Changes the default mapping between physical PCI slot numbers and logical slot numbers. • Changes the default resource assignments for slots. • Modifies the PCI configuration space polling rate. • Modifies the software disconnection timeout. • Specifies alternate HS_CSR drivers.
Slot Information	Retrieves current information about boards installed in a CompactPCI system.

To access a utility, select **Start**->**Programs**->**Hot Swap Kit** and select the utility in the menu. For more information about a specific utility, start the utility and press F1.

Configuring Hot Swap under UNIX

To allow hot-swapping of boards in your CompactPCI UNIX system, adequate address space must be preconfigured. To maximize the number of slots available for hot swapping:

- Have all slots populated at boot time, or
- Have no slots populated at boot time.

To learn more about how to allocate space for hot swapping on UNIX systems, refer to Configuring PCI bus address space for Hot Swap.

Note: For information about configuring NMS Hot Swap under Linux, refer to the Pigeon Point Systems web site (http://www.pigeonpoint.com) for information about the Pigeon Point Hot Swap Kit, HSK, and HSK kernel.

Determining PCI bus and slot locations

To configure the boards under NMS OAM, determine the logical CompactPCI or PCI bus and slot information for each NMS board installed in the system. After the boards are configured, you can identify each board using methods described in *Board identification methods* on page 14.

The *pciscan* utility displays the logical CompactPCI or PCI bus and slot information for each NMS board installed in the system. To determine the PCI bus and slot numbers for each board:

1. Access a command prompt on the target host system.

   If the host system is a remote host, access the command prompt on the host using a separate third-party utility such as *telnet*, *rsh*, or *rexec*.

2. Run *pciscan* by entering:

   ```
   pciscan
   ```

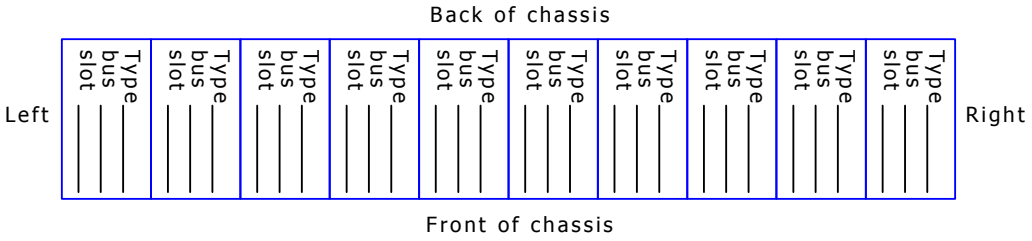
   The pciscan output is similar to the following:

   ```
   Bus Slot NMS ID
   --- ---- ------
   2    11  0x50d  AG_4000C_E1
   2    13 0x6000  CG_6000C_QUAD
   --- ---- -- -----------------
   There were 2 NMS PCI board(s) detected
   ```

3. Record the PCI bus and slot numbers.

You can also use *pciscan* to flash an LED on a specific board. For more information, refer to *Board locate: pciscan* on page 79.

A chart like the following is useful when mapping out the chassis:



CompactPCI or PCI chassis mapping

Specifying PCI bus numbers for board search functions

If you have a chassis with an unusual PCI bus topology (for example, bus number 171 directly follows bus number 0), utilities that search the PCI buses for boards and other information function more slowly. These utilities include:

- *pciscan*
- *oamcfg*'s board auto-detect function described in *Automatically detecting and adding boards* on page 61.
- *oamgen* described in *System configuration file creator: oamgen* on page 77.

To get around this problem, create a text file listing the buses that these utilities must search.

The text file must be named *nmspcinfo.cfg*. It must be located either in the *\nms\ctaccess\cfg* directory (*/opt/nms/ctaccess/cfg* under UNIX) or in the local directory from which *pciscan*, *oamgen*, and *ctdaemon* are launched.

In the file, PCI bus numbers must appear on one line, separated by semicolons and preceded by the keyword Bus=. Bus numbers must appear in numerical order. For example:

```
Bus=0;1;2;120;169
```

Specify a range of consecutive bus numbers as shown here:

```
Bus=0..3;6..120;169
```

Configuring the H.100 or H.110 bus clock

If the boards are connected to each other on the H.100 or H.110 bus, set up a bus clock to synchronize communications between the boards connected to the bus. In addition, to provide redundant and fault-tolerant clocking between devices on the bus, configure alternative (fallback) clock sources to provide the clock signal if the primary source fails.

To configure the bus clock for a system:

- Configure a board to act as clock master, driving the bus clock.
- (Optional) Configure another board to act as secondary clock master, driving the clock if the primary clock master fails.
- Configure primary and secondary timing references for each clock master board. The timing reference for a board is an external signal from which it can derive a clock pulse.
- Configure all other boards as clock slaves, so they synchronize to the clock master signal.

For information about clocking, see *CT bus clocking overview* on page 87. For specifics on setting up clocking for the boards, refer to your board documentation.

Clock Management EMC

The NMS OAM service provides H.100 and H.110 bus clock management services to boards in a chassis that is connected through the bus. This functionality is provided in the Clock Management EMC.

When the boards in a system are started, the Clock Management EMC:

- Configures the clock on each board as specified in the NMS OAM database.
- Ensures that the bus clock master board (the board driving the clock) is running before setting the clocks on the slave boards.

For more information about the Clock Management EMC, refer to the *NMS OAM Service Developer's Reference Manual*.

4 Starting NMS OAM

Starting Hot Swap

This topic describes how to start Hot Swap under the following operating system environments:

- Windows
- UNIX

Starting Hot Swap under Windows

When Natural Access is installed on a host, the Hot Swap manager is installed as a Windows service. It is configured to be started manually, as follows:

1. Access a command prompt on the host system.
2. Enter:

```
net start hsmgr
```

To set the Hot Swap manager to start automatically on a host using the Windows Control Panel, follow these steps:

1. Open the **Administrative Tools** applet in the Control Panel.
   The Administrative Tools window appears.
2. Open the **Services** applet within this window.
   The Services window appears.
3. Double-click on **NMS HotSwap Manager**.
   The Properties window appears.
4. In the Startup Type drop-down menu, select **Automatic**.
5. Click **OK**.
6. Close the Services window.
7. Close the Administrative Tools window.

For more information, refer to *Hot Swap manager: hsmgr* on page 70.

Starting Hot Swap under UNIX

When Natural Access is installed, the Hot Swap driver and Hot Swap manager are placed in the */opt/nms/hotswap/bin* directory. These services can be started as daemons or as console applications.

Note: The Hot Swap manager requires the LD_LIBRARY_PATH environment variable to be set to LD_LIBRARY_PATH = /opt/nms/lib:/opt/nms/hotswap/lib.

To start the Hot Swap applications in console mode:

1. Access a command prompt on the host system.
   If the host system is a remote host, access the command prompt on the host using a separate third-party utility such as *telnet*, *rsh*, or *rexec*.
2. Start the Hot Swap driver by entering:

   ```
   /opt/nms/hotswap/bin/hssrv
   ```

3. Start the Hot Swap manager by entering:

   ```
   /opt/nms/hotswap/bin/hsmgr
   ```

   This script sets the LD_LIBRARY_PATH environment variable and starts the Hot Swap manager in console mode.

To start the Hot Swap applications as daemons:

1. Access a command prompt on the host system.
   If the host system is a remote host, access the command prompt on the host using a separate third-party utility such as *telnet*, *rsh*, or *rexec*.
2. Start the Hot Swap driver in daemon mode by entering:

   ```
   /opt/nms/hotswap/bin/hssrv -d
   ```

3. Ensure that the LD_LIBRARY_PATH environment variable is set to LD_LIBRARY_PATH = /opt/nms/lib:/opt/nms/hotswap/lib.
4. Start the Hot Swap manager in daemon mode by entering:

   ```
   /opt/nms/hotswap/bin/hsmgr -d
   ```

To run the services in daemon mode at boot time (recommended), edit the */etc/inittab* file to include lines that set the LD_LIBRARY_PATH environment variable and then start the Hot Swap driver and Hot Swap manager. In this case, do not include the -d command line option. For more information about the *inittab* file, refer to the UNIX administrator manuals.

For more information, refer to *Hot Swap driver: hssrv* on page 75 and *Hot Swap manager: hsmgr* on page 70.

Starting the Natural Access Server

Before you use NMS OAM or any related utility, start the Natural Access Server (*ctdaemon*) on each resource host. *ctdaemon* must be running for NMS OAM functions to be available.

This topic provides the following information about starting Natural Access:

- Starting Natural Access under Windows
- Starting Natural Access under UNIX
- Starting the Natural Access Server in the in-process mode

Note: If *ctdaemon* is stopped, all dependent applications receive an error. Stop and restart the service for NMS OAM functions to become available again. Note that applications accessing Natural Access in in-process mode only are not affected if *ctdaemon* is shut down.

In order for the NMS OAM Supervisor to start up within the Natural Access Server when it boots, the following line must appear in the [ctasys] section in *cta.cfg* (this line is included by default):

```
Service = oam, oammgr
```

Starting the Natural Access Server under Windows

Under Windows, start *ctdaemon* as a service using a console window, or in the Control Panel.

To start *ctdaemon* in a console window:

1. Access a command prompt on the host system.
   If the host system is a remote host, access the command prompt on the host using a separate third-party utility such as *telnet*, *rsh*, or *rexec*.
2. Enter the following:
   ```
   net start ctdaemon
   ```

To start *ctdaemon* using the Control Panel:

1. Open the **Administrative Tools** applet in the control panel.
   The Administrative Tools window appears.
2. Open the **Services** applet within this window.
   The Services window appears.
3. Double-click on **NMS ctdaemon**.
   The Properties window appears.
4. Click **Start**.
5. Click **OK**.
6. Close the Services window.
7. Close the Administrative Tools window.

For console interaction with the NMS *ctdaemon* service, invoke ctdaemon -c from any command prompt while the service is running.

Starting the Natural Access Server under UNIX

On UNIX systems, invoke ctdaemon -i from the command prompt. This method allows full console interaction with the *ctdaemon*.

Starting the Natural Access Server in the in-process mode

In certain debugging scenarios, it is useful to start the Natural Access Server in the in-process mode. When the Natural Access Server runs in this mode, tracing messages are reported directly to stdout.

To start the Natural Access Server in the in-process mode:

1. Access a command prompt on the host system.
   If the host system is a remote host, access the command prompt on the host using a separate third-party utility such as *telnet*, *rsh*, or *rexec*.
2. Enter the following:

```
ctdaemon
```

When the Natural Access Server is not running in the in-process mode, tracing messages are captured in *agpierror.log*. Under Windows, this file is located in *\nms\oam\log*. Under UNIX it is located in */opt/nms/oam/log*. Use the *dectrace* utility to decode ISDN information from this file, as follows:

```
dectrace -f \nms\oam\log\agpierror.log > mytrace.txt
```

Verifying Hot Swap

Once you have started the Hot Swap manager and driver on a host, use *oammon* to verify that all Hot Swap files are installed and the Hot Swap driver and the Hot Swap manager are running. To verify the Hot Swap installation:

1. Create records for the components in the system in the NMS OAM database, as described in *Configuration file overview* on page 41.
2. Access a command prompt on the host system.
3. Start *oammon* by entering:

```
oammon
```

4. If you open the ejector handles on a CompactPCI board, messages reporting the extraction display. For example:

```
Wed Sep 26 13:55:18 - HSWEVN_REMOVAL_REQUESTED INFO Board 0 "Name0"
HotSwap notification
Wed Sep 26 13:55:18 - HSWEVN_BOARD_OFFLINE INFO Board 0 "Name0"
HotSwap notification
```

5. When you physically remove the board, the following message displays:

```
Wed Sep 26 13:55:57 - HSWEVN_BOARD_REMOVED INFO Board 0 "Name0"
HotSwap notification
```

6. If you insert a CompactPCI board when board auto-start is disabled (see *Starting boards automatically* on page 51), messages reporting the insertion display, but the board fails to start. For example:

```
Wed Sep 26 13:57:11 - HSWEVN_BOARD_INSERTED INFO Board 0 "Name0"
HotSwap notification
Wed Sep 26 13:57:11 - HSWEVN_ONLINE_PENDING INFO Board 0 "Name0"
HotSwap notification
Wed Sep 26 13:57:11 - HSWEVN_PREPARATION_FAILED INFO Board 0 "Name0"
HotSwap notification

Later, when the board is started, a Hot Swap message reporting the board start is
displayed:

Wed Sep 26 13:54:10 - HSWEVN_BOARD_READY INFO Board 0 "Name0"
HotSwap notification
```

7. If you insert a CompactPCI board when board auto-start is enabled (see *Starting boards automatically* on page 51), messages reporting the insertion and the board start display. For example:

```
Wed Sep 26 13:57:11 - HSWEVN_BOARD_INSERTED INFO Board 0 "Name0"
HotSwap notification
Wed Sep 26 13:54:10 - HSWEVN_ONLINE_PENDING INFO Board 0 "Name0"
HotSwap notification
Wed Sep 26 13:54:10 - HSWEVN_BOARD_READY INFO Board 0 "Name0"
HotSwap notification
```

For more information, see *Using oammon* on page 54.

Logging startup events

NMS OAM automatically maintains the following NMS OAM event logs on each host:

File name	Description
startup.log	A list of all NMS OAM events that occurred when the host started. This file is rewritten whenever the host starts. It is closed after the host starts.
oam.rpt	Low-level report information generated whenever a board is started. Information about each board is appended to the file when the board is started. The file is rewritten when the host is started. **Note:** For users migrating from *agmon*: this file is the NMS OAM equivalent of the *ag.rpt* file generated by *agmon*.

On the host, the files are found in the following locations:

- Windows: *\nms\oam\log*
- UNIX: */opt/nms/oam/log*

Configuring PCI bus address space for Hot Swap (UNIX only)

To allow hot-swapping of boards in your CompactPCI UNIX system, adequate address space must be preconfigured. To maximize the number of slots available for hot-swapping:

- Have all slots populated at boot time, or
- Have no slots populated at boot time.

This topic describes how to allocate space for hot-swapping.

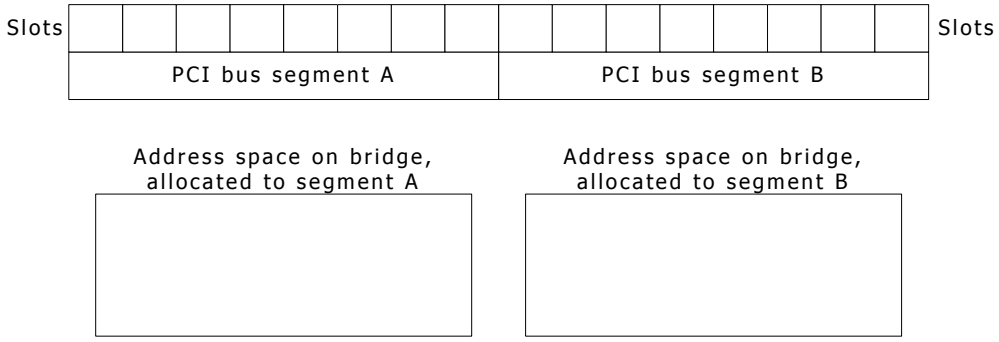
Note: Windows system address space is configured automatically during installation of the Hot Swap Kit.

PCI bus segments and space windows

The PCI architecture allows a system to include a tree of PCI buses. Most CompactPCI systems have at least two PCI bus segments: one on the processor board and one or more dedicated to CompactPCI slots. There is at least one bus segment per 8 CompactPCI slots. PCI-to-PCI bridges connect these buses.

PCI bus slots and segments

Each device requires a certain amount of address space on the bridges. At boot time, the system BIOS configures address space windows on each bridge to define the range of addresses (that is, the bus number or memory address) that are allocated behind that bridge.



Segments and allocated address space on bridge

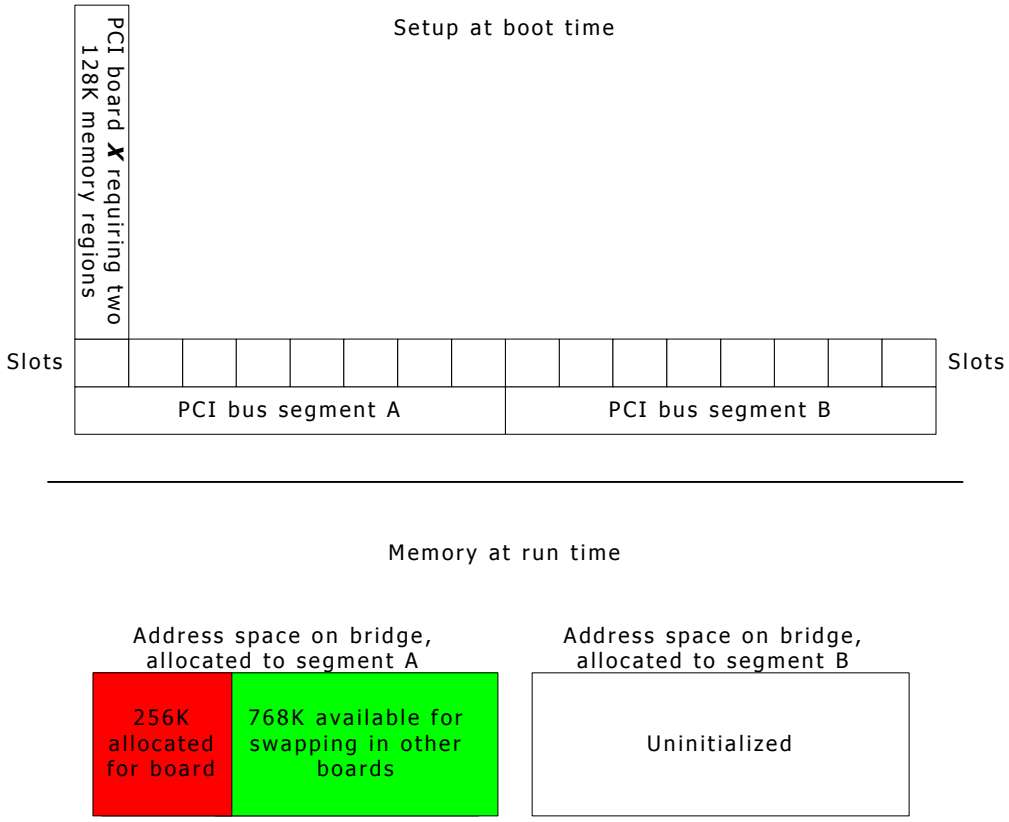
Boards can be hot-inserted only into slots for which memory has been preallocated. Memory is usually allocated as follows:

- If any devices are physically installed at boot time, the bridge windows are initialized to be just big enough to span the address spaces that have been allocated to these devices behind the bridge. In this case, boards can be hot-inserted only into slots that were populated at boot time. (This is true unless the boards can fit into leftover allocated space, as described below.)
- If no devices are physically installed at boot time, a single large bridge window is initialized that can accommodate any number of boards that can fit into it. This window is 16 MB under Windows; 64 MB under UNIX.

Thus to maximize the number of slots available for hot-swapping, you should have all slots populated at boot time or have no slots populated at boot time.

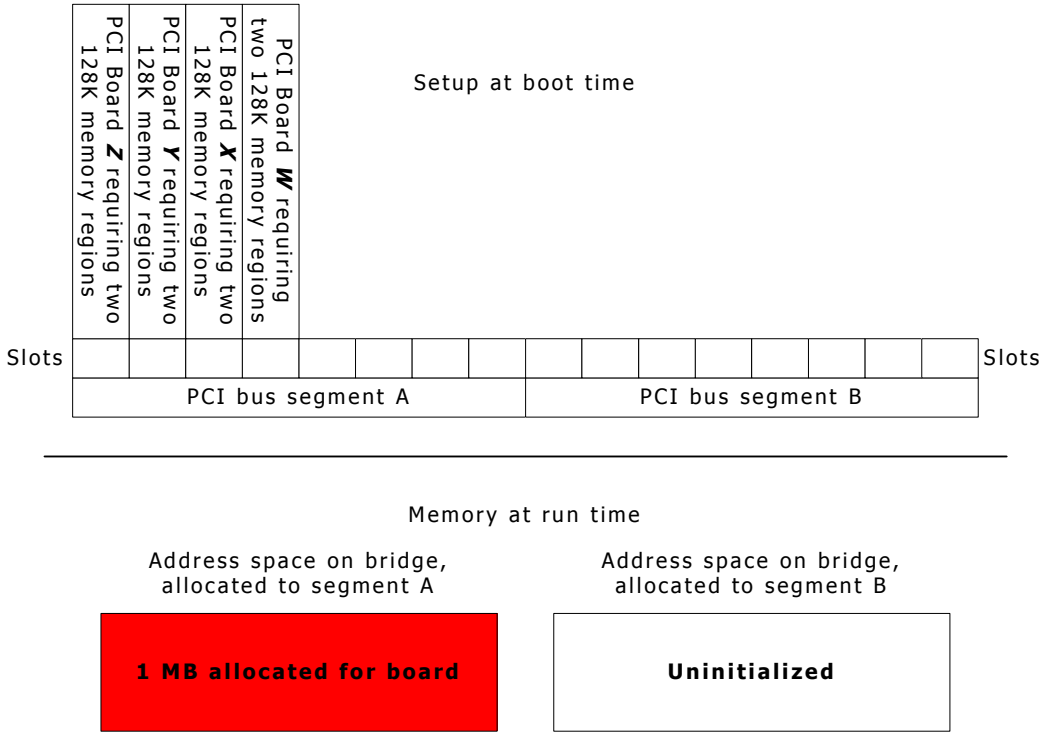
Using leftover allocated space

Usually, each address space window cannot be less than 1 MB in size. If allocations to boards behind the bridge do not add up to an integral number of megabytes, some fraction of a megabyte will be available in the window and unallocated. This unallocated space is then available for insertion of additional boards whose address space requirements are small enough. For example, if a board requires two 128K memory regions, and a CompactPCI bus segment contains only one of these boards at boot time, hot-insertion of up to three additional boards into that segment can be accommodated.



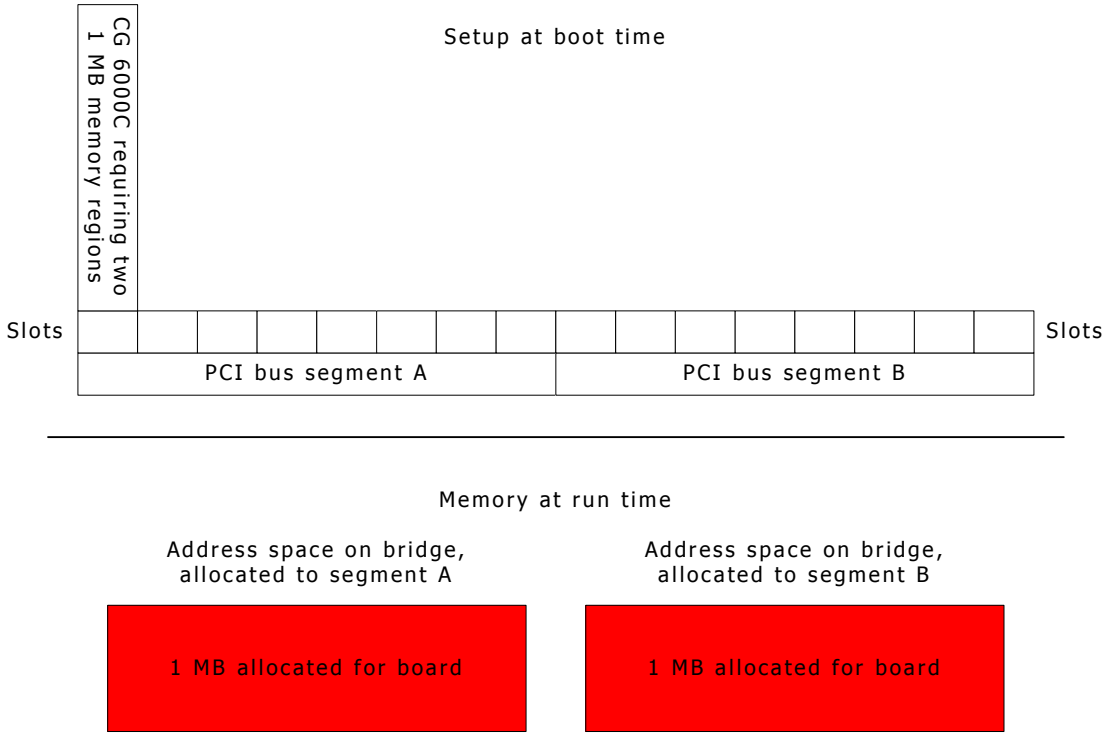
Bus with 256K board inserted

However, if an 8-slot segment has four slots occupied at boot time with the boards, no more boards can be hot-inserted into that segment because four boards occupy exactly one megabyte of address space.



Bus with four 256K boards inserted

Some boards (such as the CG 6000C board) have an address space requirement of two 1 MB memory regions. Since this requirement exactly matches the 1 MB granularity, you cannot add more of these boards than were present at start-up without rebooting.



Bus with CG 6000C board inserted

5 Creating NMS OAM configuration files

Configuration file overview

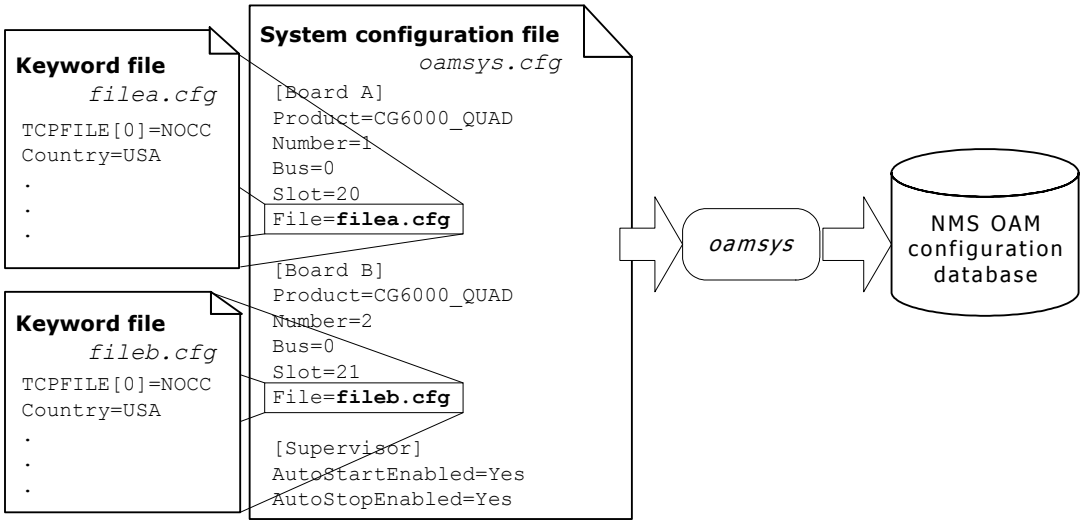
Once you have determined the internal layout of the system, create NMS OAM configuration files describing the layout. Then run *oamsys* to initialize the NMS OAM database based on the information in the file.

To set up NMS OAM, create a system configuration file. This file contains:

- A list of boards in the system.
- For each board, the name of one or more board keyword files containing keywords and values to configure the board. These settings are expressed as keyword name and value pairs.

  You can also include sections to configure non-board components, such as an EMC or the Supervisor. For more information, see *Configuring non-board objects* on page 44.

When *oamsys* runs, a record is created for each object in the NMS OAM database, containing default parameter settings. Then the settings in the configuration files are added to the record.



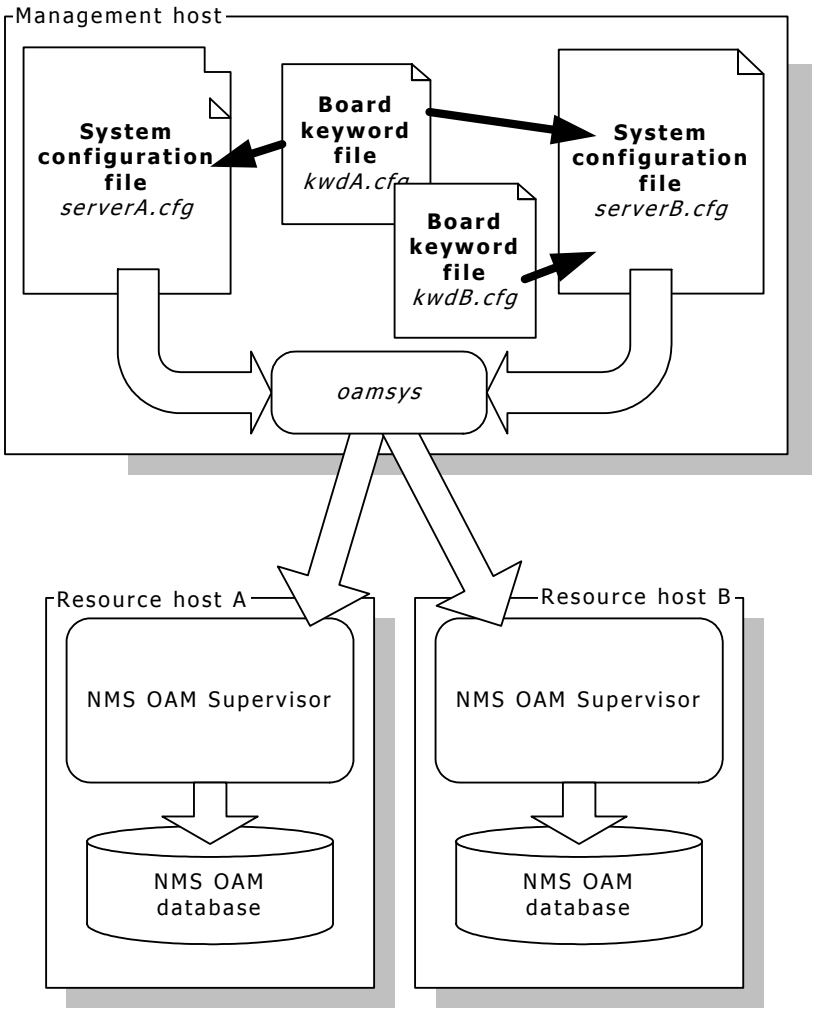
NMS OAM configuration files

If your system contains more than one board with the same configuration, you can use the same keyword file for each of these boards.

Several sample keyword files are supplied with the hardware installation. Each of these files configures the board to use a different protocol (for example, wink start or off-premises station). You can reference these files in the system configuration file or modify them. For more information about the sample files supplied for the hardware, refer to the board documentation.

System configuration files in multiple-host configurations

If you have a multiple-host NMS OAM configuration, create a separate system configuration file for each resource host, containing configuration information for that host only. Store all system configuration files and board keyword files on the host system. To configure and start up resource hosts, run *oamsys* multiple times, each time specifying a different resource host and system configuration file. The system configuration files can share keyword files if necessary.



Using oamsys in multiple-host configuration

If the resource hosts have identical configurations, you can use the same system configuration file for each resource host. However, having boards with the same board name on more than one resource host can lead to confusion.

Creating a system configuration file

A system configuration file is an ASCII text file. Typically, this file is named *oamsys.cfg*. By default, *oamsys* looks for a file with this name when it starts up.

If you know the PCI bus and slot locations of the boards in your system, create the system configuration file manually as described in this topic. If you do not know the locations of your boards, use the *oamgen* utility (included with the NMS OAM software) to create a skeleton system configuration file for your system. You can then complete this file manually, and then run *oamsys*. For more information about *oamgen*, refer to *System configuration file creator: oamgen* on page 77.

This topic provides the following information:

- Configuration file sections
- Mandatory statements
- Board keyword sections
- Configuring non-board objects
- Sample configuration file

You can find a sample system configuration file in the *nms\oam\cfg* directory under Windows or in */opt/nms/oam/cfg/* under UNIX.

Configuration file sections

Statements within the system configuration file appear one to a line. Any text appearing after a number sign (#) is a comment and is ignored. Statements in all configuration files are not case sensitive, except where operating system conventions prevail (for example, file names under UNIX).

The system configuration file is divided into multiple sections, one for each board. Each section is headed with the name of the board, in square brackets ([ ]):

```
[Myboard]
```

This name can contain a space (for example: [My board]), but must not be preceded by or followed by a space. For example, the following names are not valid:

```
[ Myboard], [Myboard ].
```

Note: Board names must be unique within a system configuration file.

Below each board name are statements that apply to the board. Each statement appears on its own line. Each statement consists of a keyword name, followed by an equal sign (=) and then a value:

keyword_name*=*value

Mandatory statements

In the section for each board, the following statements must appear:

Keyword	Description
Product	Name of the board product. To learn how to retrieve a list of valid strings to use, see *Displaying board product types* on page 60.
Number	Board number. Use any integer from 0 to 32767. Each board's number must be unique.
Bus	PCI bus number. The bus:slot location for each board must be unique.
Slot	PCI slot number. The bus:slot location for each board must be unique.

Board keyword sections

To specify a keyword file for the board, use the File keyword:

```
File = myfile.cfg
```

You can specify more than one keyword file. Specify the file names on a single line following the File keyword, separated by a space:

```
File = file1.cfg file2.cfg file3.cfg
```

Alternatively, you can specify multiple File keywords, one to a line:

```
File = file1.cfg
File = file2.cfg
File = file3.cfg
```

To include embedded spaces in a file name, surround the name with quotation marks:

```
File = "My Configuration File.cfg"
```

By default, *oamsys* searches for the keyword files listed with this keyword in the same way it searches for the system configuration file itself (see *Running oamsys* on page 53). To reference a file in another directory, specify the directory along with the file name:

```
File = c:\mydir\file1.cfg
```

Keywords are set in the order in which *oamsys* encounters them in the files. Specifying a setting for a keyword in more than one file is not recommended.

Note: In addition to (or instead of) keyword file names, you can specify keyword settings for a board directly in the board's section in the system configuration file. Use the keyword syntax described in *Specifying keywords and values* on page 48.

Configuring non-board objects

In addition to sections for boards, the system configuration file can include sections containing configuration information for non-board objects (such as EMCs, board plug-ins, or the NMS OAM Supervisor).

The section for each object is headed with the object's name, in square brackets ([ ]):

```
[Supervisor]
```

The object name for the NMS OAM Supervisor is Supervisor. The object name for a plug-in or EMC is its file name (for example, *hotswap.emc*).

This name must not be preceded by or followed by spaces. For example, the following names are not valid:

```
[ Supervisor], [Supervisor ]
```

Below each board name are keyword settings, specified as described in *Specifying keywords and values* on page 48. For example:

```
[Supervisor]
AutoStartEnabled=Yes
AutoStopEnabled=Yes
```

The File statement can also be used to specify a keyword file containing settings for the object:

```
[Supervisor]
File=supvparms.cfg
```

To learn what keywords can be set for board plug-ins, refer to the board-specific documentation. To learn what keywords can be set for EMCs or the NMS OAM Supervisor, refer to the *NMS OAM Service Developer's Reference Manual*.

Sample configuration file

The following system configuration file describes two CG 6000C boards, one at PCI bus 0, slot 20, and the other at PCI bus 0, slot 21. The first board is assigned keyword file *a6wnk.cfg*, which sets up the board to use the wink start protocol. The second board uses keyword file *a6ops.cfg*, which sets up the board to use the off premises station protocol. Supervisor keywords are set to start the boards automatically when the system boots or when they are Hot Swap inserted, and to stop automatically when the system shuts down:

```
# This is the NMS OAM system configuration file.
# It describes all the boards in my system.

[My board]
Product = CG_6000C_QUAD
Number = 1
Bus = 0
Slot = 20
File = 6wnk.cfg #Wink Start protocol

[My other board]
Product = CG 6000C QUAD
Number = 2
Bus = 0
Slot = 21
File = 6ops.cfg #Off Premises Station protocol

[Supervisor]
AutoStartEnabled=Yes
AutoStopEnabled=Yes
```

To start boards automatically when the Supervisor starts up, set the AutoStart keyword for each board. For more information about this keyword, refer to the *NMS OAM Service Developer's Reference Manual*.

Using board keyword files

A board keyword file contains keyword settings. When you create the system configuration file, you can reference one or more board keyword files to use for the components in your system (see *Creating a system configuration file* on page 43). When you run *oamsys*, the utility adds the settings for each component to the NMS OAM database.

Several sample keyword files are supplied with the hardware installation. Each of these files configures the board to use a different protocol (for example, wink start or off-premises station). You can reference these files in the system configuration file or modify them. For more information about the sample files supplied for your hardware, refer to the board documentation.

For detailed descriptions of the keywords supported for the board, refer to the board documentation.

If the system contains more than one board with the same configuration, you can use the same keyword file for each of these boards.

Note: All sample files set each board to stand-alone clocking mode. For boards to communicate with each other across the CT bus, modify the clocking information for each board as described in *CT bus clocking overview* on page 87.

Keyword file syntax

A keyword file is an ASCII text file. Typically, the file has the extension *.cfg*.

Within the file, each statement appears on its own line. A line beginning with a number sign (#) denotes a comment and is ignored. If a line ends with a backslash (\), the next line is assumed to be a continuation of the line.

Board keyword file example

The following board keyword file configures a CG 6000C board to run with NOCC. Board-specific information (such as board ID information) is not included in board keyword files.

```
#
#     c6nocc.cfg
#     CG 6000 configuration file
#
#     This file configures the board to run Voice with NOCC.
#



Clocking.HBus.ClockMode                          = STANDALONE
Clocking.HBus.ClockSource                        = OSC
Clocking.HBus.ClockSourceNetwork                 = 1
TCPFiles                                         = nocc
DSPStream.VoiceIdleCode[0..3]                    = 0x7F
DSPStream.SignalIdleCode[0..3]                   = 0x00
NetworkInterface.T1E1[0..3].Type                 = T1
NetworkInterface.T1E1[0..3].Impedance            = DSX1
NetworkInterface.T1E1[0..3].LineCode             = B8ZS
NetworkInterface.T1E1[0..3].FrameType            = ESF
NetworkInterface.T1E1[0..3].SignalingType        = CAS
DSP.C5x[0..31].Libs[0]                           = cg6klibu
DSP.C5x[0..31].XLaw                              = MU LAW
DSP.C5x[1..31].Files                             = voice tone dtmf echo \
                                                   rvoice callp ptf wave \
                                                   oki ima gsm ms g726 mf
DSP.C5x[0].Files                                 = qtsignal tone dtmf echo \
                                                   callp NULL NULL
Resource[0].Name                                 = RSC1
Resource[0].Size                                 = 120
Resource[0].TCPs                                 = nocc
#################################################################
# Before modifying this resource definition string refer to the CG6000C
# Installation and Developers Manual.
#################################################################
Resource[0].Definitions         = ( dtmf.det all & echo.ln20 apt25 & \
ptf.det 2f & tone.gen & callp.gnc & ptf.det 4f & ( (rvoice.rec mulaw & \
rvoice.play mulaw) | (rvoice.rec alaw & rvoice.play alaw) | \
(rvoice.rec lin & rvoice.play lin) | (voice.rec 16 & (voice.play 16 100 | \
voice.play_16_150 | voice.play_16_200)) | (voice.rec_24 & \
(voice.play_24_100 | voice.play_24_150 | voice.play_24_200)) | \
(voice.rec 32 & (voice.play 32 100 | voice.play 32 150 | \
voice.play 32 200)) | (voice.rec 64 & (voice.play 64 100 | \
voice.play 64 150 | voice.play 64 200)) | (wave.rec 11 16b & \
wave.play_11_16b) | (wave.rec_11_8b & wave.play_11_8b) | (oki.rec_24 & \
(oki.play_24_100 | oki.play_24_150 | oki.play_24_200)) | (oki.rec_32 & \
(oki.play 32 100 | oki.play 32 150 | oki.play 32 200)) | (ima.rec 24 & \
ima.play 24) | (ima.rec 32 & ima.play 32) | (gsm ms.frgsm rec & \
gsm ms.frgsm play) | g726.rec 32 | g726.play 32) )
DLMFiles[0]                                      = cg6krun
DebugMask                                        = 0x0
```

Specifying keywords and values

This topic provides the following information:

- Keyword name/value pairs
- Struct keywords
- Array keywords
- Array keyword expansion

Keyword name/value pairs

In its simplest form, a statement consists of a keyword name, followed by an equal sign (=) and then a value:

keyword_name = ***value***

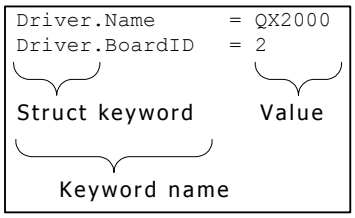
where ***keyword_name*** denotes a parameter and ***value*** indicates the value to assign the parameter:

```
AutoStart = YES
```

For a list of valid keywords for a component, see the manual for the device you are configuring. NMS OAM Supervisor keywords, Clock Management EMC keywords, and Hot Swap EMC keywords are listed in the *NMS OAM Service Developer's Reference Manual*.

Struct keywords

Struct keywords are similar to C language structures. A struct is a group of related named values (elements) under a common name. The fully qualified keyword name for each element in the struct consists of the struct name, followed by a period (.) and then the element name. Within NMS OAM, the fully qualified keyword name for an element is always used to refer to the element.



Struct keyword names

Structs can contain structs. In the following example, struct Clocking contains structs Hbus and MVIP:

```
Clocking.HBus.ClockMode = MASTER_A
Clocking.HBus.AutoFallBack = YES
Clocking.MVIP.ClockRef = SEC8K
Clocking.MVIP.AutoFallBack = NO
```

In this example, Clocking, Hbus, and MVIP are struct keywords.

Array keywords

Many keywords are organized into arrays: lists of items of the same type. Each element of the array can have a unique value.

The index for an array keyword appears as a suffix, surrounded by square brackets. Each index is zero based:

```
TCPFile[0] = nocc
```

A struct can contain arrays:

```
DSPStream.SignalIdleCode[0] = 0x00
DSPStream.VoiceIdleCode[0] = 0x00
DSPStream.SignalIdleCode[1] = 0x00
DSPStream.VoiceIdleCode[1] = 0x00
```

It is also possible to have an array of structs:

```
Resource[0].Name = RSC1
Resource[0].Size = 120
Resource[0].FileName[0] = myfile.foo
Resource[0].FileName[1] = myfile2.foo
Resource[0].SpanEnable=AUTO
Resource[1].Name = RSC1
Resource[1].Size = 60
Resource[1].FileName[0] = myfile.foo
Resource[1].SpanEnable=AUTO
```

For any array keyword ***xxx***, ***xxx***.Count indicates the number of elements in the array. For example:

```
Resource.Count=2
```

xxx.Count is automatically updated for each element added or removed from an array. This value cannot be set directly.

Array keyword expansion

For convenience, there is a shorthand method of assigning values to keywords in an array.

Multiple keyword names can be assigned the same value in a single line, as follows:

Statement	Expanded equivalent
keyword[0..2] = value	keyword[0] = value keyword[1] = value keyword[2] = value
keyword[0-2] = value	(same as previous row)
keyword[1,3,5] = value	keyword[1] = value keyword[3] = value keyword[5] = value
keyword[0..3,5..7,9] = value	keyword[0] = value keyword[1] = value keyword[2] = value keyword[3] = value keyword[5] = value keyword[6] = value keyword[7] = value keyword[9] = value

A separate range can be specified for each keyword array index in the name:

Statement	Expanded equivalent
kywd1[1].kywd2[1..2] = value	kywd1[1].kywd2[1] = value kywd1[1].kywd2[2] = value
kywd1[1..3].kywd2[1..2] = value	kywd1[1].kywd2[1] = value kywd1[1].kywd2[2] = value kywd1[2].kywd2[1] = value kywd1[2].kywd2[2] = value kywd1[3].kywd2[1] = value kywd1[3].kywd2[2] = value

Multiple values for keywords in an array can be specified on a single line, separated by spaces. To include spaces in a value, surround the value with quotation marks. Values are assigned to keywords in numerical order, starting with 0. The array keyword is specified without the square brackets or index value (for example, Resource for Resource[***x***]):

Statement	Expanded equivalent
keyword = val1 val2 val1 val4	keyword[0] = val1 keyword[1] = val2 keyword[2] = val1 keyword[3] = val4
keyword = val1 val2 "val 1" val4	keyword[0] = val1 keyword[1] = val2 keyword[2] = "val 1" keyword[3] = val4
kywd1[1..3].kywd2[1..2].list = val1 val2	kywd1[1].kywd2[1].list[0] = val1 kywd1[1].kywd2[1].list[1] = val2 kywd1[1].kywd2[2].list[0] = val1 kywd1[1].kywd2[2].list[1] = val2 kywd1[2].kywd2[1].list[0] = val1 kywd1[2].kywd2[1].list[1] = val2 kywd1[2].kywd2[2].list[0] = val1 kywd1[2].kywd2[2].list[1] = val2 kywd1[3].kywd2[1].list[0] = val1 kywd1[3].kywd2[1].list[1] = val2 kywd1[3].kywd2[2].list[0] = val1 kywd1[3].kywd2[2].list[1] = val2

Note: For users of the NMS OAM service API: *oamcfg* performs keyword expansion, not NMS OAM. When specifying keywords and values using the NMS OAM service, do not use this keyword expansion syntax.

Starting boards automatically

Using keywords, you can configure each board to start automatically whenever the Natural Access Server (*ctdaemon*) starts, or when the board is Hot Swap inserted. You can also configure the boards to stop automatically whenever *ctdaemon* exits.

Note: When a board is Hot Swap extracted, it is stopped automatically regardless of the keyword settings. The board is stopped when its ejector handles are lifted.

To configure boards to start or stop automatically:

1. For each board that you want to start or stop automatically, set the AutoStart or AutoStop keywords to Yes.
   AutoStart and AutoStop are set to No by default.
2. Set the Supervisor keywords AutoStartEnabled or AutoStopEnabled to Yes. These keywords enable auto-starting or auto-stopping of all boards whose AutoStart or AutoStop keywords are set to Yes.
   You can set Supervisor keywords directly in the system configuration file, as shown in the following example:

```
[Supervisor]
AutoStartEnabled = Yes
AutoStopEnabled = Yes
```

Consider the following information when automatically starting and stopping boards:

- Your application must not attempt to access a board before its start process is completed. An application can determine that a board is completely started by monitoring for OAM board start done events. To learn more about NMS OAM events, refer to the *NMS OAM Service Developer's Reference Manual*.
- When boards are started, the Clock Management EMC or a clock management program such as *clockdemo* (described in *Running clockdemo* on page 99) can alter the state of the board's clocking based upon board configurations. As a result, certain clock-dependent functionality is not immediately available to an application following the board start done event. To determine when functionality is available, an application can receive and interpret clocking events.
- If *ctdaemon* is stopped while an application is running, the boards are no longer accessible using NMS OAM. NMS OAM board events are not reported to the application. Also, the application can receive unexpected service API errors.

6 Using oamsys and oammon

Using oamsys

Use the *oamsys* utility to set up the NMS OAM database, based upon parameter values specified in a system configuration file. This utility:

- Stops any currently operating boards.
- Creates records for board components in the NMS OAM database based on a system configuration file you supply. Any existing board-specific data in the database is deleted and replaced with the contents of the system configuration file. For more information about system configuration files, see *Configuration file overview* on page 41.
- Configures non-board components as described in *Configuring non-board objects* on page 44.
- Attempts to start all boards.

To perform its tasks, the *oamsys* utility makes multiple calls to the *oamcfg* utility, described in *oamcfg overview* on page 57.

To use *oamsys*, *ctdaemon* must be running. To learn how to start Natural Access in this mode, refer to *Starting the Natural Access server* on page 33.

To configure and start up boards on multiple systems, run *oamsys* once for each system, specifying the target system on the command line. *oamsys* can configure only one system at a time.

Running oamsys

To launch *oamsys*, enter:

```
oamsys [options]
```

where ***options*** are:

Option	Use this option to...
-f ***filename***	Specify the file name (and path, if necessary) of a system configuration file to load. If you invoke *oamsys* without this option, it searches for a file named *oamsys.cfg* in the current directory, and then in the paths specified in the AGLOAD environment variable. If you specify a file name without an extension, *oamsys* assumes the extension to be *.cfg.*
-@ ***host***	Load the configuration file on resource host ***host***. ***host*** is an IP address or machine name. If unspecified, the operation is performed on the host on which the utility was initialized.

oamsys reads system configuration files, not board keyword files. Board keyword files to be added to the NMS OAM database must be specified within the system configuration file. Refer to *Creating a system configuration file* on page 43.

When invoked with a valid file name, *oamsys* does the following:

- Checks the syntax of the system configuration file and checks that all required keywords are present. *oamsys* reports all syntax errors it finds.

  Note: *oamsys* checks syntax only on the system configuration file and not on any keyword files referenced in the file.

- Checks for uniqueness of board name, number and bus/slot within the system configuration file.
- Shuts down all boards referenced in the NMS OAM database (if any).
- Deletes all board configuration information currently stored in the NMS OAM database (if any).
- Sets up the NMS OAM database according to the specifications in the system configuration file.
- Attempts to start all boards, as described in the database.

oamsys invokes *oamcfg* repeatedly to perform its actions. With each invocation, the command line is displayed. For more information about *oamcfg*, see *oamcfg overview* on page 57.

Using oammon

The *oammon* utility enables you to perform the following operations:

- Monitor for board errors and other messages from the OAM system
- Capture these messages in a flat file (*oammon.log*)
- Send an alert notification message to all NMS OAM client applications

Note: NMS OAM utilities such as *oammon* can only run if *ctdaemon* and NMS OAM have been started. Therefore *oammon* does not log OAM startup information. OAM startup information is logged to the file *startup.log* (described in *Logging startup events* on page 36).

To launch *oammon*, enter oammon at the command line, followed by any command line options. Precede each option with a hyphen (-). If the option includes data, specify the data directly after the option on the command line. Valid options are described in *Command line options* on page 56.

When you invoke *oammon* without command line options, *oammon* displays an interactive menu and immediately begins logging messages to a file named *oammon.log.* The *oammon* interactive menu appears as follows:

```
  'Enter' to output log file tail, based on
   current screen output line count
'c'     to change screen output line count
   (current count is 10)
'p'     to poll screen output every 10 seconds
'q'     or 'x' to exit
```

Enter any of the following commands:

Command	Enter this command to...
c	Change the number of lines of message that *oammon* prints to the screen every 10 seconds (the default is ten lines). After entering c, enter a positive integer to indicate the number of lines of message text to print.
p	Print the last 10 lines of received message text to the screen every 10 seconds. This is a toggle command. The first time you enter p, it enables on-screen polling. The second time you enter p, it disables on-screen polling.
q	Exit *oammon*.

For *oammon* to report messages, *ctdaemon* must be running. (To learn how to start Natural Access in this mode, refer to *Starting the Natural Access Server* on page 33.) If *oammon* is started before *ctdaemon*, it displays:

```
Waiting for CT Access Server...
```

When *oammon* is running and *ctdaemon* starts, *oammon* displays the interactive menu and begins logging messages to the file *oammon.log (*located in *\nms\oam\log* under Windows, and */opt/nms/oam/log/* under UNIX or Linux). Each time you start *oammon*, the previous *oammon.log* file is moved to *oammon.bak*, and a new *oammon.log* file is created for the current session.

Messages reported by *oammon* include trace messages from managed components in the system. For more information about tracing, refer to the board documentation.

To monitor multiple hosts, start a separate instance of *oammon* for each host. Each instance monitors one host only.

Command line options

The following table describes the *oammon* command line options:

Option	Use this option to...
-?	Display a Help screen and terminate.
-h	Display a Help screen and terminate.
-v	Run in verbose mode and return extended board information.
-s ***messagetext***	Send a test alert notification message containing text ***messagetext*** to all applications currently monitoring for alert messages (for example, another instance of *oammon* that is monitoring). *oammon* then terminates. ***messagetext*** can be any string of characters. Applications receive an OAMEVN_ALERT event containing a pointer to an OAM_MSG structure containing the message text. For more information about alert notification, refer to the *NMS OAM Service Developer's Reference Manual*.
-L	Print logged messages to stdout. This option is recommended for diagnostic purposes only.
-f ***filename***	Log messages to a file named ***filename*** and to stdout. This option is ignored unless the -L option is also used.
-@ ***server***	Monitor activity on a resource server, where ***server*** is a host name or IP address. If unspecified, *oammon* monitors the local host on which it was initialized. This option is ignored unless the -L option is also used.

7 Using oamcfg

oamcfg overview

The NMS OAM configuration utility *oamcfg* enables you to perform the following operations:

- Create or delete boards in the database
- Specify settings for a component's parameters, either individually or collectively, using board keyword files
- Start or stop one or more boards
- Test boards (if supported)
- Detect boards in a system
- Display basic ID information for boards
- Import or export the contents of the OAM database

oamcfg can perform a given operation for a single board, or it can configure all boards in a single invocation.

Launching oamcfg

To launch *oamcfg*, enter oamcfg on the command line, followed by zero or more command line options. Precede each option with a hyphen (-) or slash (/). If the option includes data, specify the data directly after the option on the command line. Valid options are described in the following table.

Note: To use *oamcfg*, *ctdaemon* must be running. To learn how to start *ctdaemon*, refer to *Starting the Natural Access Server* on page 33.

Command line options

Use the -b, -g, -l, -m, and/or -n options to specify a board or other component for the operations. If you do not specify a board or component with these options, the specified operations are performed for all boards.

Option	Use this option to...
-@ ***host***	Perform the operation on resource host ***host***. ***host*** is an IP address or machine name. If unspecified, the operation is performed on the host on which the utility was initialized.
-a	Create a record in the NMS OAM database for a board that was automatically detected using the -x option. The board name is specified with the -n option. If the board name is omitted, all detected boards are added. For details, see *Automatically detecting and adding boards* on page 61. When this option is used, all other options (except the -x option) are ignored.
-b ***brdno***	Specify the target board number. If this option and the -g, -l, -m, and -n options are omitted, any specified operations are performed for all boards. Use this option to change the board number. For details, see *Changing board ID information* on page 64.
-c ***product***	Create a record in the NMS OAM database for the board containing basic board ID information. ***product*** is the product string for the board type. If ***product*** is ?, *oamcfg* displays a list of all board product types supported by the installed plug-ins, in alphabetical order, and then terminates. If ***product*** is" ", *oamcfg* chooses the first product name in this list.
-d	Delete the records for the boards from the NMS OAM database.
-export ***filename***	Export a snapshot of the contents of the NMS OAM database to ***filename***. The contents of the file can then be imported into another NMS OAM database by invoking oamcfg with the -import option.
-f ***cfgfile***	Add the information from keyword file ***cfgfile*** to the database records for the specified components. This option can appear more than once on a command line, to load multiple files. Statements in the board keyword file override information already in the record. **Note:** *oamcfg* is designed to parse keyword files, not system configuration files such as those that *oamsys* takes as input.
-g ***shelf***:***slot***	Specify the shelf and slot of the board for which to perform the specified operation. If this option and the -b, -l, -m, and -n options are omitted, the specified operation is performed for all boards.
-h or ?	Display *oamcfg* Help screen and terminate.
-i	Cause *oamcfg* to return immediately when used with the -p, -s, and -t options. By default, *oamcfg* does not return until it receives indications that its operations have completed (successfully or not).

Option	Use this option to...
-import ***filename***	Import the contents of an NMS OAM database from ***filename***. All current records are destroyed and replaced with those from the file.
-k ***keyword***=***value***	Set ***keyword*** to ***value*** in the database record for the specified component. This option can appear more than once on a command line, to set multiple keywords.
-l ***PCIbus***:***slot***	Specify the target board by PCI bus and slot. If this option and the -b, -g, -m, and -n options are omitted, any specified operation is performed for all boards. Use this option to change the PCI bus and slot location specified in the database for a board. For details, see *Changing board ID information* on page 64.
-m ***MAC_address***	Specify the target board by one of its MAC addresses. If this option and the -b, -g, -l, and -n options are omitted, any specified operation is performed for all boards.
-n ***name***	Specify the target board or component by name. ***name*** can be the name of a board or another component (such as, an EMC or the Supervisor). If this option and the -b, -g, -l, and -m options are omitted, the specified operation is performed for all boards.
-p	Stop the specified boards. The board stops immediately, interrupting any ongoing process. To avoid problems, make sure a board is not performing any operations before stopping it.
-q	Query the NMS OAM database for the board ID information for the specified boards.
-r	Cause *oamcfg* to reset all keywords to their default values (except board ID information) for the specified components. *oamcfg* then makes the specified changes. Use when configuration data in the NMS OAM database is being changed (that is, the -f or -k option is used or board ID information is changed). If the -r option is omitted, *oamcfg* adds or replaces keyword values specified in the board keyword file without disturbing any other settings.
-s	Start the specified boards.
-t ***testopts***	Test the specified boards, if supported by the board plug-in. ***testopts*** is a numeric value indicating how to perform the test.
-x	Search the chassis for boards and adds information about each board it finds into subkeywords of the DetectedBoards[] Supervisor array. Use the -a option to create records for detected boards in the NMS OAM database. When this option is used, all other options (except the -a option) are ignored.

A single invocation of *oamcfg* can perform multiple operations by specifying more than one operation on the command line. For example, the following command line creates a record in the NMS OAM database for a CG 6000C board in bus 0, slot 20, displays the board's ID parameters, loads keyword file *cgnocc.cfg* (replacing all existing information, if any) and attempts to start the board:

```
oamcfg -l 0:20 -c CG_6000C_QUAD -q -f cgnocc.cfg -r -s
```

Identifying boards in oamcfg operations

To indicate the board on which *oamcfg* is to perform a specified operation, use any of the following options:

Option	Use this option to identify a board by...
-b ***boardno***	Its board number.
-g ***shelf:slot***	Its shelf and slot location. Only valid for boards in CompactPCI chassis with PICMG 2.1-compliant buses.
-l ***PCIbus*:*slot***	Its PCI bus and slot location.
-m ***MACaddr***	The MAC address of one of its Ethernet chips. Only valid for boards with Ethernet capability.
-n ***name***	Its board name.

For example, the following command queries for information about the board in PCI bus 1, slot 14:

```
oamcfg -l 1:14 -q
```

If you omit a board identification option from the command line, *oamcfg* performs the specified operation for all boards. For example, the following command queries for information about all boards:

```
oamcfg -q
```

To perform an *oamcfg* operation for a board on a resource host, use the -@ command line option to specify the target host. For example, the following command queries for information about the board in PCI bus 1, slot 14, on resource host IP_3:

```
oamcfg -@ IP_3 -l 1:14 -q
```

Displaying board product types

When specifying board configuration information in a system configuration file, you must supply the product type for each board. The product type is a string that identifies the board type to NMS OAM.

Different board plug-ins support different board types. To determine what strings to specify for your boards, you can query NMS OAM for the board types supported by the installed plug-ins. To do so, enter:

```
oamcfg -c?
```

oamcfg returns a list of supported board types, in alphabetical order. Each listed product type is a valid string that you can use to identify your products in the system configuration file.

Adding and deleting boards

This topic describes how to perform the following tasks with *oamcfg*:

- Create a record in the database.
- Automatically detect and add boards.
- Delete a board.

Creating a record in the database

To create a record in the NMS OAM database for the object, enter:

```
oamcfg -c product [-l PCIbus:slot] [-n brdname] [-b brdno] [-@ host]
```

The following table explains each option:

Option	Use this option to specify...
-c ***product***	Product string for the board type. Refer to *Displaying board product types* on page 60 for information on retrieving a list of valid product name strings. If ***product*** is omitted *oamcfg* chooses the first product name in this list.
-l ***PCIbus:slot***	PCI bus and slot location of a board in the system. If this option is omitted, *oamcfg* assumes PCI bus 0, slot 0.
-n ***brdname***	Name to give the board. If this option is omitted, a default name is generated.
-b ***brdno***	Number to give the board (0 - 15). If this option is omitted, a default number is generated.
-@ ***host***	Machine name or IP address of the resource host where the board is located. If not specified, *oamcfg* performs its operations on the host on which it is initialized.

The -g and -m options cannot be used to identify the board in this operation. Refer to *Command line options* on page 56.

Note: The board must currently be physically installed in the system for its name to be added or deleted from the NMS OAM database.

For example, the following command adds a record for a CG 6000C board located in PCI bus 0, slot 20 on resource host MyHost1:

```
oamcfg -c CG_6000C_QUAD -l 0:20 -@ MyHost1
```

When a record is created for a board, the record includes a unique name and board number for the board. You can use either of these identifiers to refer to the board in future calls. To learn how to retrieve this information, see *Reading and changing database information* on page 63.

You can change the board name or number. For details, see *Changing board ID information* on page 64.

Automatically detecting and adding boards

oamcfg can automatically detect boards in a resource host chassis. You can then use *oamcfg* to add records for detected boards to the NMS OAM database on that host.

To detect boards in a chassis, use the -x option. To detect boards in a resource host, also include the -@ option indicating the host:

```
oamcfg -x -@ MyHost1
```

oamcfg detects boards in the chassis and creates entries in the DetectedBoards[x] array keyword for each board. These keywords are displayed on the screen, as follows:

```
DetectedBoards[0].Name = AG 2000 0 16
DetectedBoards[0].Product = AG 2000
DetectedBoards[0].Location.PCI.Bus = 0
DetectedBoards[0].Location.PCI.Slot = 16
DetectedBoards[1].Name = AG_2000_BRI_0_17
DetectedBoards[1].Product = AG_2000_BRI
DetectedBoards[1].Location.PCI.Bus = 0
DetectedBoards[1].Location.PCI.Slot = 17
DetectedBoards[2].Name = AG 4000 E1 0 15
DetectedBoards[2].Product = AG_4000_E1
DetectedBoards[2].Location.PCI.Bus = 0
DetectedBoards[2].Location.PCI.Slot = 15
```

Note: If you have a chassis with an unusual PCI bus topology (for example, bus number 171 directly follows bus number 0), *oamcfg*'s search functions more slowly. To speed up operation, create a text file specifying bus numbers to search. For details, refer to *Specifying PCI bus numbers for board search functions* on page 29.

To add a detected board to the NMS OAM database, use the -a option. Also specify the -n option indicating the name of the board to add. For example, the following command line adds detected board AG_2000_0_16 to the database on resource host MyHost1:

```
oamcfg -a -n AG_2000_0_16 -@ MyHost1
```

To add all detected boards at once, omit any specific board name, as follows:

```
oamcfg -a -@ MyHost1
```

Deleting a board

To delete a board from the NMS OAM database, use the -d option. The following command deletes the board named myboard:

```
oamcfg -d -n myboard
```

Note: This operation does not require that the board be physically removed from the system.

If no board is specified, *oamcfg* deletes all boards from the NMS OAM database.

Reading and changing database information

When a record is created for a board in the NMS OAM database, it is assigned a unique name and board number. You can display and change this information with *oamcfg*.

This topic describes how to use *oamcfg* to:

- Display board ID information.
- Specify settings in board keyword files.
- Specify keyword settings on the command line.
- Change board ID information.
- Replace existing data in the database.

Displaying board ID information

To display the ID parameters for a board, use the -q option. For example, the following command displays all ID parameters in the database:

```
oamcfg -q
```

oamcfg responds with output like the following:

```
NAME               NUMBER BUS:SLOT SHELF:SLOT    MAC ADDRESS    PRODUCT
AG 4040C E1 2 13      0     2:13       N/A    00-20-22-ff-13-40 AG 4040C E1
CG 6000C Quad 2 10    3     2:10      31:7    00-20-22-40-08-74 CG 6000C Quad
CG_6000C_Quad_2_11    4     2:11      31:6    00-20-22-31-19-12 CG_6000C_Quad
```

You can change the board name or number. For details, see *Changing board ID information* on page 64.

To specify keyword settings with *oamcfg*, you can:

- Supply the keywords in a keyword file. The information is stored in the NMS OAM database.
- Specify the keywords on the *oamcfg* command line.

Specifying settings in board keyword files

Use the oamcfg -f ***fname*** option to specify a board keyword file. ***fname*** is the name of a board keyword file. You can include this option more than once, to specify more than one file. If no board is specified, *oamcfg* loads the keyword file for all boards.

The following command adds the configuration information in keyword files *filea.cfg* and *fileb.cfg* to the database record for board 1:

```
oamcfg -b 1 -f filea.cfg -f fileb.cfg
```

If you omit the path, *oamcfg* searches for the specified files in the current directory, and then the paths specified in the AGLOAD environment variable.

To search elsewhere, specify the entire path along with the file name on the command line.

If you specify a file name without an extension, *oamcfg* assumes the extension to be *.cfg*.

To specify a space within a file name, surround it with quotation marks:

```
oamcfg -b 1 -f "My File.cfg"
```

Specifying settings on the command line

To set a specific keyword, you can specify it directly on the command line using the -k ***keyword***=***value*** option. ***keyword*** is a valid keyword name for the component, and ***value*** is a valid value for the keyword.

The keyword and value must be separated by an equal sign (=). For example:

```
oamcfg -b 1 -k DebugLevel=3
```

If you need to embed a space in a keyword and value designation, place the whole designation in quotation marks. For example:

```
oamcfg -b 1 -k "DebugLevel = 3"
```

The -k option can appear more than once on a command line, to set multiple values.

If no board is specified, *oamcfg* sets the keyword for all boards in the NMS OAM database.

For more information about keywords and values, see *Using board keyword files* on page 46.

Changing board ID information

You can change the number or PCI bus and slot information for a board. To do so, specify a board on the command line using a board identification option (-b, -g, -l, -m, or -n). This board must be currently listed in the database. Specify the new number, PCI bus and slot, or both using other -b or -l options on the same command line.

oamcfg checks the database for each option. If it determines that only one option specifies current information for an existing board, it assigns that board the number, PCI bus:slot, or both given in the other options.

To change the number of the board in bus 0, slot 20, specify the following (assumes that board number 5 does not currently exist):

```
oamcfg -l 0:20 -n myboard -b 5
```

Note: You cannot change a board's name, MAC address, or shelf and slot information.

You cannot specify the same board identification option twice on the same command line. When referencing an existing board with a given identification option, you must specify two command lines to change that option. For example, to change board number 0 to 15 (assuming that board number 15 does not currently exist), specify the following:

```
oamcfg -b 0 -n temp
oamcfg -n temp -b 15
```

Replacing existing data

By default, when *oamcfg* adds, changes, or deletes information for a component (using the -f or -k options), or changes board ID information (as described in *Changing board ID information* on page 64), it does not disturb any other settings for the board. Use the -r option to delete all database information for the board before adding the new information. This is useful when you want to start from a blank slate when changing information for a component:

```
oamcfg -b 1 -r -f filea.cfg -f fileb.cfg
```

Starting, stopping, and testing boards

This topic provides information about performing the following tasks with *oamcfg*:

- Starting boards
- Stopping boards
- Testing boards

Starting boards

Once a board is properly configured and is physically installed in the system, use the -s option to start the board:

```
oamcfg -s -n myboard
```

If no board is specified, *oamcfg* attempts to start all boards in the NMS OAM database.

By default, *oamcfg* waits after attempting to start the specified boards until all board start attempts succeed or fail, reporting the results to stdout. To avoid this, use the -i option:

```
oamcfg -s -i
```

If the -i option is used, results are still available. The results come asynchronously encapsulated in NMS OAM events, which *oammon* can receive and display.

Stopping boards

To stop a board, use the -p option:

```
oamcfg -p -n myboard
```

If no board is specified, *oamcfg* attempts to stop all boards in the NMS OAM database.

Note: The specified board stops immediately, interrupting any ongoing process. To avoid problems, make sure a board is not performing any operations before stopping it.

By default, *oamcfg* waits after attempting to stop the board until all board stop attempts succeed or fail, reporting the results to stdout. To avoid this, use the -i option:

```
oamcfg -p -i
```

If the -i option is used, results are still available. The results come asynchronously encapsulated in NMS OAM events, which *oammon* can receive and display.

Testing boards

To test a board, use the -t ***testopts*** option:

```
oamcfg -t 0x80000301 -n myboard
```

testopts is a bit mask indicating how the test is performed. If no board is specified, *oamcfg* attempts to start testing on all boards in the NMS OAM database. Testing is started on the boards in numerical order (of board numbers).

Note: Currently, only CG boards support testing. Testing can interrupt current board activities. For this reason, do not use a board for any other operations during testing.

After attempting to start the board tests, *oamcfg* waits by default until all board test start attempts succeed or fail, reporting results to stdout and *oammon*. To avoid this wait, use the -i option:

```
oamcfg -n myboard -t 0x80000301 -i
```

If the -i option is used, results are still available. The results come asynchronously encapsulated in NMS OAM events, which *oammon* can receive and display.

Importing and exporting configurations

You can export the contents of the NMS OAM database to a file, and then import the file into the NMS OAM database on another system. This feature is useful for setting up multiple identical systems.

Note: Only the entire contents of a database can be exported. Importing a database file completely obliterates and replaces all data in an existing database.

To export the contents of a configuration database, invoke *oamcfg* with the -export ***filename*** option, where ***filename*** is the output file to create:

```
oamcfg -export myfile.cfg
```

oamcfg exports a snapshot of the entire configuration to the specified output file.

Note: Do not modify the output file. It is for use by NMS OAM only.

To import the contents of a configuration database from a file, invoke *oamcfg* with the -import ***filename*** option, where ***filename*** is the file to import:

```
oamcfg -import myfile.cfg
```

oamcfg imports the entire configuration from the specified input file. The current configuration is lost and is replaced by the new configuration. All plug-ins are restarted.

oamcfg task sequence

Regardless of the order in which the options are specified, *oamcfg* always performs operations in the following order:

1. If -x is specified, ignores all other command line options except -a. Searches the system for boards.
2. If -a is specified, ignores all other command line options except -x. Adds any detected boards to the NMS OAM database.
3. If -c is specified, creates a board in the NMS OAM database unless -c? is specified. If -c? is specified, *oamcfg* displays a list of all product types supported by the installed plug-ins, in alphabetical order, and then terminates.
4. Assigns the board a default name, number, PCI bus, and slot. The following defaults are used:

Item	Default
Name	Product name, followed by a space and then a numeral. For example: `CG 6000C QUAD 0`
Number	Next unused number. For example, if board 1 exists, the next number is board 2.
PCI bus	0
PCI slot	0

Note: For each operation (except -c), if no specific component is referenced on the command line with the -b, -l, or -n options, the operation is performed for all boards on the resource host specified with the -@ option, or on the management host if -@ is not specified.

5. Assigns board ID information if specified on the command line. Values specified on the command line override any values previously set.

Note: If the -r option is specified, any existing data for the boards is deleted when any new information is added with the -f or -k options, or if the board ID information changes (as described in *Reading and changing database information* on page 63).

6. In the NMS OAM database record(s) for the components, adds the contents of any keyword files specified with -f options.
7. In the NMS OAM database record(s) for the components, sets any values specified with -k options on the command line.
   The value for a given keyword specified on the command line overrides any value for that keyword previously loaded from a keyword file.
8. If -q is specified, displays the board's name and number, or the names and numbers of all boards if no board is specified on the command line.
9. If -s is specified, attempts to start the board, or all boards if no board is specified on the command line.
   By default, *oamcfg* waits until all board start or test attempts succeed or fail, unless the -i option is specified.
10. If -p is specified, stops the boards.
11. If -t is specified, tests the boards.
12. If -d is specified, deletes the boards from the NMS OAM database.

8 Other utilities

Utilities overview

In addition to the configuration and monitoring utilities *oamsys*, *oammon*, and *oamcfg*, the following utilities are available with Natural Access software:

Program	Description
hsmgr	Runs the Hot Swap manager.
hsmon	Monitors the Hot Swap manager.
hssrv	Starts and coordinates the set of Hot Swap drivers (UNIX only).
oamgen	Creates a system configuration file.
pciscan	Determines PCI and CompactPCI bus and slot locations.
trunkmon	Displays the status of digital trunks.
blocate	For UNIX systems, associates the PCI bus assignment to a physical board by flashing an LED on the board's end bracket.

Hot Swap manager: hsmgr

Name

hsmgr

Purpose

Runs the Hot Swap manager.

Usage

`hsmgr` [***options***]

where ***options*** are:

Option	Description
-c	(Windows only) Starts the Hot Swap manager as a console application.
-d	(UNIX only) Starts the Hot Swap manager as a daemon.
-h, -?	Displays the *hsmgr* Help screen and terminates.
-k	(UNIX only) Kills previous instance of the daemon.
-n	Disables display of messages and states.
-o ***log_file***	Specifies an output log file for messages instead of writing to standard output.

Description

hsmgr must be running to use Hot Swap. When Natural Access is installed, *hsmgr* is installed as a service and is configured to be started manually.

To run on a remote host, the utility must be physically resident on the remote host. Use a separate third-party utility such as *telnet*, *rsh*, or *rexec* to invoke the utility.

When debugging Hot Swap applications, run *hsmgr* in console mode to see Hot Swap manager messages.

Procedure

To run *hsmgr* in console mode:

1. To stop any previous instance of *hsmgr*:

   Under Windows:

   a. Open the **Administrative Tools** applet in the Control Panel.

      The Administrative Tools window appears.

   b. Open the **Services** applet within this window.

      The Services window appears.

   c. Double-click on **NMS HotSwap Manager**.

      The Properties window appears.

   d. Click **Stop**.

   e. Click **OK**.

   f. Close the Services window.

   g. Close the Administrative Tools window.

   Under UNIX:

   Run the Hot Swap manager with the -k option to stop any previous instance of the manager:

   ```
   hsmgr -k
   ```

2. Start the Hot Swap manager in console mode by entering:

   ```
   hsmgr -c
   ```

   Note: In UNIX, if you are running the Hot Swap manager in console mode, ensure that the Hot Swap driver (*hssvr*) is running, otherwise startup fails.

If the print option is on (default), messages display as boards are inserted and extracted. Each message displays in the following format:

direction destination pci_bus, pci_slot hsmessage

where:

Field	Description
direction	Direction of message: > indicates an output message < indicates an input message.
destination	Label given to an application (for example, hsmon) or the label for querying a board (for example, QSlotI).
pci_bus, ***pci_slot***	CompactPCI bus and slot location.
hsmessage	Hot Swap manager message indicating the Hot Swap state or message.

For example:

```
>QSlotI   0,9 HSM REPLY SLOT BY IDENT DATA
<QSlotI   0,0 HSM OPEN CONNECTION
<QSlotI   0,0 HSM QUERY SLOT BY IDENT DATA
>QSlotI   0,9 HSM_REPLY_SLOT_BY_IDENT_DATA
<QSlotI   0,0 HSM_CLOSE_CONNECTION
<QState   0,0 HSM OPEN CONNECTION
<QState   0,9 HSM QUERY HSM STATE
>QState   0,9 HSM REPLY HSM STATE  status HSMS P0
<QState   0,0 HSM CLOSE CONNECTION
<OAM      0,0 HSM_OPEN_CONNECTION
<OAM      0,0 HSM_CLOSE_CONNECTION
<HSMON    0,0 HSM OPEN CONNECTION
<HSMON    0,0 HSM OPEN CONNECTION
<HSMON    0,9 HSM QUERY HSM STATE
>HSMON    0,9 HSM_REPLY_HSM_STATE  status HSMS_P0
<HSMON    0,0 HSM_CLOSE_CONNECTION
<HSMON    0,0 HSM OPEN CONNECTION
<HSMON    0,9 HSM QUERY SLOT INFO
>HSMON    0,9 HSM REPLY SLOT INFO
<HSMON    0,0 HSM_CLOSE_CONNECTION
```

Error messages

hsmgr displays the following error messages:

Error message	Description
Error: Can't create 'hsmgr_hsd' event object	Hot Swap manager cannot create the *hsmgr_hsd* event object. Check system resources.
Error: Can't create 'hsmgr_hsf' event object	Hot Swap manager cannot create the *hsmgr_hsf* event object. Check system resources.
HSMgr: initialization error	This message usually follows other error messages. Check to see if another copy of the Hot Swap manager is running.
pci bus, slot HSMgr internal error: Wrong transition from ***old state*** to ***new state***	Hot Swap manager encountered an error transitioning between states.
pci bus, slot Skipped HSM_BOARD_CONFIGURED message	A board preparation application sent an unexpected message.

Informational messages

hsmgr displays the following informational messages:

Informational message	Description
Use ***statediagram*** diagram	On startup, the Hot Swap manager displays the state diagram it is using.
Changed from ***oldstatediagram*** to ***newstatediagram*** diagram	If the state diagram changes, the Hot Swap manager displays the new diagram information.

Hot Swap monitor: hsmon

Name

hsmon

Purpose

Monitors the Hot Swap manager.

Usage

`hsmon` [***options***]

where ***options*** are:

Option	Description
s	Stops *hsmon*.
i ***PCIbus***,***slot***	Inserts a board. Initiates management-driven insertion.
e ***PCIbus***,***slot***	Extracts a board. Initiates management-driven extraction.
g ***PCIbus***,***slot***	Gets the state of the specified slot.
q	Terminates *hsmon*.
?	Displays the *hsmon* Help screen and terminates.

Description

hsmon traces all messages from the Hot Swap manager. Use this utility for installation verification and diagnostics.

To run on a remote host, the utility must be physically resident on the remote host. Use a separate third-party utility such as *telnet*, *rsh*, or *rexec* to invoke the utility.

Procedure

1. Make sure the Hot Swap manager and Hot Swap driver are running.
2. To launch the Hot Swap monitor, enter:

   ```
   hsmon
   ```

   Hot Swap manager messages display in this format:

   direction destination pci_bus, pci_slot hsmessage

   where:

Field	Description
direction	Indicates the direction of the message: > indicates an output message < indicates an input message
destination	Label given to an application (for example, hsmon) or the label for querying a board (for example, QSlotI).
pci_bus, ***pci_slot***	CompactPCI bus and slot location.
hsmessage	Hot Swap manager message indicating the Hot Swap state or message.

3. Insert a board. The following messages display:

   ```
   < 1,12 HSM_BOARD_CONFIGURED
   < 1,12 HSM_S0_S1 Board is configured
   < 1,12 HSM_S1_S1I Device instance is created
   < 1,12 HSM_PREPARE_BOARD
   < 1,12 HSM_S1I_S1B Board preparation requested
   < 1,12 HSM_S1B_S2 Board is ready
   < 1,12 HSM_BOARD_READY
   ```

4. Enter s to stop the Hot Swap monitor. The following messages display:

   ```
   Stopping monitor...
   monitor stopped.
   ```

5. Enter q to quit.

Hot Swap driver: hssvr

Name

hssrv

Purpose

Starts and coordinates the set of Hot Swap drivers (UNIX only).

Usage

`hssrv` [***options***]

where ***options*** are:

Option	Description
-h, -?	Displays the *hssrv* Help screen and terminates.
-mc	Prints configuration related messages.
-mi	Prints information messages.
-me	Prints warnings and error messages.
-ma	Prints all messages.
-c	Starts the Hot Swap driver as a console application (default).
-d	Starts the Hot Swap driver as a daemon.
-k	Kills any previous instance of the daemon.

Description

On a UNIX system, *hssrv* must be running to use Hot Swap. When Natural Access is installed, *hssrv* is placed in the */opt/nms/hotswap/bin* directory. Start *hssrv* as a daemon or as a console application. To run *hssrv* at boot time (recommended), add information about the program to the */etc/inittab* file. For more information, see the UNIX administrator manual.

To run on a remote host, the utility must be physically resident on the remote host. Use a separate third-party utility such as *telnet*, *rsh*, or *rexec* to invoke the utility.

When debugging Hot Swap applications, use *hssrv* in console mode (the default) to see Hot Swap driver messages.

Procedure

To run *hssrv* in console mode:

1. Stop NMS OAM and any Natural Access applications.
2. Stop *hsmgr*.
3. Invoke *hssrv* with the option -k to stop any previous instance of the driver:

   ```
   hssrv -k
   ```

4. Reboot the system.
5. Start *hssrv* in console mode by entering:

   ```
   hssrv -c
   ```

If a print option is included on the command line (-m***message_type***), messages display as boards are inserted and extracted.

There are three types of messages:

Configuration messages (messages related to a device configuration process)

```
hssrv: EXT ACK (1:9:0) -> S0E
hssrv: Remove 40100000-4011FFFF
hssrv: Remove 40120000-4013FFFF
hssrv: Connected through bridge (0:8)
hssrv: BASE 0 32 bit - 128.00 KB - Configure as 40100000-4011FFFF
hssrv: BASE 1 32 bit - 128.00 KB - Configure as 40120000-4013FFFF
hssrv: Assign IRQ for (1: 9)
hssrv: RT (2) - (0:5:0)
hssrv: IRQ10 configured.
hssrv: aghw - [AG PCI Board]
```

Error and warning messages

```
hssrv: Device is not in RT table.
hssrv: Warning - SetHWInt is not supported
hssrv:         - Assuming that IRQ is preconfigured
```

Informational messages

```
hssrv: - hsbios (PCI BIOS Interface)             - Loaded.
hssrv: - hsrmgr (Resource Manager Interface)     - Loaded.
hssrv: - hshw   (CompactPCI Hardware Interface)  - Loaded.
hssrv: PCI BIOS found. 3 bus(es)
hssrv: IRQ routing table - 9 record(s)
hssrv: Check for reserved resource manager keys
hssrv: - 14 reserved key(s)
hssrv: Get current system configuration
hssrv: PCI IDE - Mark IRQ14 (Primary channel is in compatibility mode)
hssrv: PCI IDE - Mark IRQ15 (Secondary channel is in compatibility mode)
hssrv: - 8 PCI device(s) were found
hssrv: - IRQs ( 7 6 8 1 4 3 10 11 5 14 5 11 10 )
hssrv: - 16.93 MB allocated by devices
hssrv: Search for PCI2PCI bridges
hssrv: - PCI2PCI bridge at (0: 8) #0 -> #1
hssrv: - Memory window - 40100000-401FFFFF,  1 MB
hssrv: - PCI2PCI bridge at (0:12) #0 -> #2
hssrv: - Memory window - 40200000-402FFFFF,  1 MB
hssrv: Shared resources 00000001 / 0000000D
hssrv: 24 Software driver(s) configured
```

System configuration file creator: oamgen

Name

oamgen

Purpose

Scans a chassis for boards and creates a system configuration file describing the board setup.

Note: The system configuration file created by *oamgen* may not be appropriate for your configuration. You may need to make further modifications to the file before running *oamsys* to configure your boards based on the file.

Usage

```
oamgen
```

This utility has no command-line options.

Description

Use *oamgen* to create a system configuration file describing the configuration of the boards in a chassis. *oamgen* creates a file called *sample.cfg*, located in the directory from which *oamgen* was invoked.

Note: For *oamgen* to operate, *ctdaemon* must be running. To learn how to start *ctdaemon*, refer to *Starting the Natural Access Server* on page 33.

As *oamgen* locates each board, it displays the product name and PCI bus:slot location of the board. For example:

```
NMS OAM configuration file generator

...Inserting board AG_4040C_E1 (2:9)
...Inserting board CG_6000C_Quad (2:10)
...Inserting board CG 6100C (2:11)
...Inserting board AG 4040C E1 (2:13)

  A sample OAM configuration file sample.cfg has been generated.
  To boot the NMS boards in the chassis, use the command:

     oamsys -f sample.cfg
```

In the system configuration file, *oamgen* assigns a board name and number to each board. Boards are numbered in the order in which *oamgen* discovers them in the system starting with board 0. The board name for each board is Name***x***, where ***x*** is the board number, for example, Name0.

oamgen also assigns each board a keyword file, based on the board's product type. Each keyword file is one of the sample keyword files installed for the boards in the *\nms\oam\cfg* directory (*opt/nms/oam/cfg* under UNIX). To learn what sample board keyword files are installed for your board types, refer to the board documentation.

When *oamgen* is complete, you can immediately run *oamsys* to configure and start the boards in the system based on *sample.cfg*. Alternatively, you can modify *sample.cfg* to suit your configuration before running *oamsys*. For more information about system configuration files, refer to *Creating a system configuration file* on page 43. For more information about oamsys, refer to *Using oamsys* on page 53.

If you have a chassis with an unusual PCI bus topology (for example, bus number 171 directly follows bus number 0), *oamgen* functions more slowly. To speed up operation, create a text file specifying PCI bus numbers to search. For details, refer to *Specifying PCI bus numbers for board search functions* on page 29.

Note: Wink start protocol keyword files are installed on your system only if one of the countries you chose during software installation is the United States. If the *sample.cfg* file output by *oamgen* calls for a wink start protocol keyword file, modify *sample.cfg* or install the wink start protocol keyword files before running *oamsys*.

The following example is a typical *sample.cfg* file generated by *oamgen*:

```
#-------------------------------------------------------
# sample.cfg
#
# Sample OAM configuration file generated by oamgen
# based on the NMS boards found in this chassis.
#
# To boot the NMS boards in this chassis, use the command:
#      oamsys -f sample.cfg
#
# You may need to modify the keyword configuration files
# on the "File = XXXX.cfg" lines to suit your needs.
#-------------------------------------------------------
[Name0]
        Product = AG 4040C E1
        Number = 0
        Bus = 2
        Slot = 9
        File = agpi4000.cfg   # AG 4040C E1, Wink-start protocol

[Name1]
        Product = CG 6000C Quad
        Number = 1
        Bus = 2
        Slot = 10
        File = c6nocc.cfg     # CG 6000C, T1, No call control
[Name2]
        Product = CG 6100C
        Number = 2
        Bus = 2
        Slot = 11
        File = c61enocc.cfg   # CG 6100C, E1, No call control


[Name3]
        Product = AG_4040C_E1
        Number = 3
        Bus = 2
        Slot = 12
        File = agpi4000.cfg   # AG 4040C E1, Wink-start protocol
```

Board locate: pciscan

Name

pciscan

Purpose

Determines the PCI bus and slot assignments for NMS PCI and CompactPCI boards installed in the system.

Usage

`pciscan` [***options***]

If you invoke *pciscan* without any command line options, it returns the locations of all NMS PCI and CompactPCI boards in the system.

If you invoke *pciscan* with command line arguments, the specified board flashes an LED.

The following table lists the valid command line options:

Option	Description
pci_bus pci_slot	Specifies the PCI bus and slot location of the board on which to flash an LED.
-h, -?	Displays the Help screen and terminates.
-a	Returns the locations for all PCI devices in the system, including NMS PCI boards.
-i	Displays additional information (CG boards only).
-l	Logs output to a file named *pci_cfg.txt*.
-r	Displays five PCI memory addresses.
-v	Displays register values for NMS boards.

Description

pciscan displays the PCI bus and PCI slot number for all NMS CompactPCI boards installed in the system.

To run this utility on a remote host, the utility must be physically resident on the remote host. Use a separate third-party utility such as *telnet*, *rsh*, or *rexec* to invoke the utility.

If you have a chassis with an unusual PCI bus topology (for example, bus number 171 directly follows bus number 0), *pciscan* functions more slowly. To speed up operation, create a text file specifying PCI bus numbers to search. For details, refer to *Specifying PCI bus numbers for board search functions* on page 29.

On Windows systems, you can use *pciscan* to show the physical slot location of a specific board by flashing an LED on the board.

Note: On UNIX systems, use *cg6ktool* to flash a board LED on a specific CG board (as described in the CG board documentation), or *blocate* to flash a board LED on a specific AG board.

Procedure

To run *pciscan*, enter:

```
pciscan
```

pciscan displays output similar to the following:

```
Bus Slot  NMS ID
--- ----  ------
  2   11   0x50d  AG 4040C E1
  2   13  0x6000  CG 6000C QUAD
--- ----  -- ----------------
There were 2 NMS PCI board(s) detected
```

If the -l option is specified, the board configuration is also logged to an ASCII text file with the current date and time. The log is created in a file named *pci_cfg.txt*, in the current working directory.

To flash an LED on a specific board under Windows, run *pciscan* with the PCI bus and PCI slot locations. For example:

```
pciscan 0 14
```

An LED on the board flashes.

If the -i option is specified, extra information is reported for CG boards only. This information includes:

- Number of DSPs on the board
- Number of HMIC switches on the board
- Number of lines (digital or analog) on the board
- Number of Ethernet chips on the board
- Whether or not the board has a daughterboard
- Number of CPUs on the board

This information is reported as follows:

```
Bus Slot  NMS ID DSP Switch Line Eth xCard CPU
--- ----  ------ --- ------ ---- --- ----- ---
  2   10  0x6000  32      1    4   2    No   1 CG_6000C_Quad
  2   11  0x6000  32      1    4   2    No   1 CG 6000C Quad
  2   13   0x50d N/A    N/A  N/A N/A   N/A N/A AG 4040C T1
--- ----  -- ----------------
There were 3 NMS PCI board(s) detected
```

If the -i option is specified and a specific PCI bus and slot are specified, detailed information is reported for the board at the specified location, as follows:

```
pciscan 2 10 -i
NMS PCI Boards Scanner


*****************************
* Board CG 6000C Quad (2:10)
*****************************

Board Standard
---------------
Family =                0x6000
TestLevel =             0x1712
TestLevelRev =          0x03
SoftwareComp =          0x00
OAMProductNo =          0x0601
MFGYear =               0x07D0
MFGWeek =               0x2C
ATETestBit =            0x00
SerialNum =             0x00056271
Reserved =              0x00000000

CG6000(C) Board Specific
---------------
AssemblyLevel =         0x1706
AssemblyRev =           0x0202
AssemblyYear =          0x07D0
AssemblyWeek =          0x24
Reserved =              0x00
FlashID =               0x01
FlashBlkSz =            0x10
NumDSPCores =           0x0020
DSPSpeed =              0x0064
DSPExtClk =             0x0014
DSPType =               0x01
CTBusType =             0x01
HostBusType =           0x03
NumSwitch =             0x01
SwitchType =            0x04
NumLine =               0x04
LineType =              0x03
NumEthernet =           0x02
EthernetType =          0x01
NumDaughterCard =       0x00
Reserved2 =             0x00
NumCPU =                0x01
CPUSpeed =              0x00E9
CPUExtClk =             0x0003
CPUType =               0x01
DRAMSz =                0x20
SRAMSz =                0x00
Reserved =              0x00
NICAddr[0] =            00.20.22.40.08.74.
NICAddr[1] =            00.20.22.40.08.75.
```

Digital trunk status: trunkmon

Name

trunkmon

Purpose

Displays the status of digital trunks.

Usage

`trunkmon` [***options***]

where ***options*** are:

Option	Description
-b ***board***	Specifies the board to monitor. Default = 0.
-s	Enables beep when trunk alarm state changes. Default = no beep.
-?	Displays the *trunkmon* Help screen and terminates.
-l ***mode***	Sets the framer loopback mode. Available ***mode*** entries include: 0 = No loopback - normal mode 1 = Line loopback 2 = Backplane loopback The loopback option works only on CG boards and AG 4040 boards.
-t ***trunknumb***	Specifies the trunk number. The trunk number is a zero-based index associated with the board by NMS OAM. If no trunk number is specified, *trunkmon* defaults to trunk 0.
`-@` ***host***	Displays the status of trunks on resource host ***host***. If unspecified, it is assumed that the trunks are on the host on which *trunkmon* was initialized.
-q	For diagnostic purposes only. Queries framer loopback mode and displays the output in *oammon*.

Description

trunkmon displays the status of all trunks connected to the specified board. *trunkmon* continuously monitors the status of the trunks and updates the display if the data changes. If the -s option is specified, *trunkmon* beeps when an alarm transition occurs.

Framer loopback diagnostic options

Use the -l and -t loopback options for diagnostic purposes on CG and AG boards with DS1 interfaces. The loopback options verify whether the board's DS1 configuration matches the configuration of its incoming lines.

Caution:	*trunkmon* loopback options are intended for diagnostic purposes only. AG and CG boards cannot process data appropriately when either loopback mode is in use. In addition, the NMS compliance certificates obtained for the CG or AG board do not apply when loopback is in use.

Use the –l option to configure loopback in one of the following modes:

Loopback mode	Description
Line	Data from the board's external DS1 interfaces is looped back to the external DS1 through the framer chip's line interface unit (LIU).
Backplane	Data from the board's PCM highway is looped back to the board's PCM highway through the framer chip's PCM input.

Note: On AG boards, set the *agtrace* trace mask to 0x1000 before implementing loopback.

For example, if you run *trunkmon* with the following arguments:

```
trunkmon -l1 -t0
```

the first trunk is set up in line loopback mode. *trunkmon* displays the following output:

```
Set loopback mode 1 on trunk 0
```

You can use the -q option to query the current loopback mode and view the loopback output with *oammon*. For example, if you run *trunkmon* with the following arguments:

```
trunkmon -q -t1
```

oammon specifies the second trunk's loopback mode in the following way:

```
Get loopback mode 2 from trunk 1
```

Procedure

To run *trunkmon* for board number 0, enter:

```
trunkmon
```

trunkmon's output differs depending upon whether the digital trunks are ISDN primary rate (PRI) or basic rate (BRI). For boards with PRI trunks, the output resembles the following:

```
Digital Trunk Monitor      NMS Communications        Ver 1.2  Jun 21 2001
(Press F3 or ESC to exit, ALT-F1 to reset)
BOARD # 0
-------------------------------------------------------------------------
Monitor start time:
                     Tue Jun 26 11:59:21 2001
         Alarms   Remote    Errored  Failed     Code      Slips    Frame
                  alarms      sec      sec   violations              sync
-------------------------------------------------------------------------

Trunk 0    RED     NONE         15       15        0         1    NoSgnl
Trunk 1    NONE    YELLOW       15       15        2         0        OK
Trunk 2    NONE    NONE          1        0        0         0        OK
Trunk 3    NONE    NONE          1        0        0         0        OK
```

The following table explains the *trunkmon* output for PRI trunks:

trunkmon display (PRI trunks)	Description
Alarm	T1 trunks: RED = Red alarm or loss of frame BLUE = Blue alarm or AIS alarm NONE = No alarm E1 trunks: AIS = All ones alarm NO_FRM = Loss of frame 16 AIS = All ones in timeslot 16 NONE = No alarm
Remote Alarm	T1 trunks: YELLOW = Remote loss of frame NONE = No alarm E1 trunks: FAULT = Remote loss of frame NO_SMF = Remote loss of signaling multiframe NONE = No alarm
Errored seconds	One second intervals containing one or more errors
Failed seconds	T1 trunks: Number of one-second intervals that are preceded by 10 consecutive failed seconds E1 trunks: Number of one-second intervals in which loss of signal occurs, out-of-frame occurs, or excessive bit error rate is detected
Code violations	Line code violations
Slips	Slips accumulator
Frame sync	OK = Proper frame synchronization to the trunk NoSgnl = Loss of signal No Frm = Loss of frame No MF = Loss of signaling multiframe NoCRCF = No CRC frame synchronization ?????? = Unknown framing error

For boards with BRI trunks, *trunkmon*'s output resembles the following:

```
 Digital Trunk Monitor     NMS Communications       Ver 1.2  Jun 21 2001
                          (Press F3 or ESC to exit, ALT-F1 to reset)
  BOARD # 0
 ------------------------------------------------------------------------
  Monitor start time:
                         Mon Jul 23 09:10:04 2001
            State    Type   Slips   Errors  Receives   Transmits  B1  B2

  ------------------------------------------------------------------------

  Trunk 0    G3        NT     0        0       12         12      1   0
  Trunk 1    NONE      ??     0        0        0          0      0   0
  Trunk 2    F7        TE     0        0       12         11      1   0
  Trunk 3    NONE      ??     0        0        0          0      0   0
```

The following table explains the *trunkmon* output for BRI trunks:

trunkmon display (BRI trunks)	**Description**
State	Activation/deactivation layer 1 state machine (ITU-T I430): F1-F8 = TE trunks G1-G4 = NT trunks NONE = All others
Type	Indicates how stack on trunk was initialized: NT = Stack initialized as NT TE = Stack initialized as TE ?? = Stack not initialized on this trunk
Slips	Slips accumulator
Errors	Errors accumulator
Receives	Indicates bytes received
Transmits	Indicates bytes transmitted
B1	Shows if there is communication currently on the B1 channel
B2	Shows if there is communication currently on the B2 channel

AG board locate: blocate

Name

blocate

Purpose

For UNIX systems, associates the PCI bus assignment to a physical board by flashing an LED on the board.

Usage

`blocate` [***options***]

where options are:

Option	Description
pci_bus pci_slot	Specifies the PCI bus and slot location of the AG board on which to flash an LED.

Description

If no options are specified, *blocate* displays the PCI bus and PCI slot number for all AG boards installed in a UNIX system. If a PCI bus and slot number is specified on the command line, *blocate* flashes an LED on the specified board. To learn which LED flashes on your board model, refer to the board documentation.

Example: Flashing all AG board LEDs

To display the PCI bus and slot numbers for all AG boards in the UNIX system, enter:

```
blocate
```

The output resembles the following:

```
Thu Jul 10 15:51:22 There was 1 NMS AG PCI card(s) detected
BUS SLOT INTERRUPT
00 14 0xf
```

The board configuration is also logged to an ASCII text file, *pci_cfg.txt*, with the current date and time. The file is created in the current working directory.

Example: Flashing a specific board LED

To flash an LED on a specific AG board in a UNIX system, enter:

```
blocate pci bus pci slot
```

where ***pci_bus*** and ***pci_slot*** are the PCI bus and PCI slot locations of the board. For example:

```
blocate 0 14
```

The following message displays:

```
Flashing LED for NMS PCI board on bus 0 slot 14
```

The LED on the specified board flashes briefly.

9 H.100 and H.110 bus clocking

CT bus clocking overview

If the boards in a system are connected to each other on the CT bus, set up a bus clock to synchronize communications between the boards connected to the bus. In addition, to provide redundant and fault-tolerant clocking on the bus, configure alternative (fallback) clock sources to provide the clock signal if the primary source fails.

The topics in this section describe H.100/H.110 clocking as described in the *ECTF H.110 Hardware Compatibility Specification: CT Bus R1.0*. Not all boards support this specification completely. For information on setting up clocking with a particular board type, refer to the board documentation.

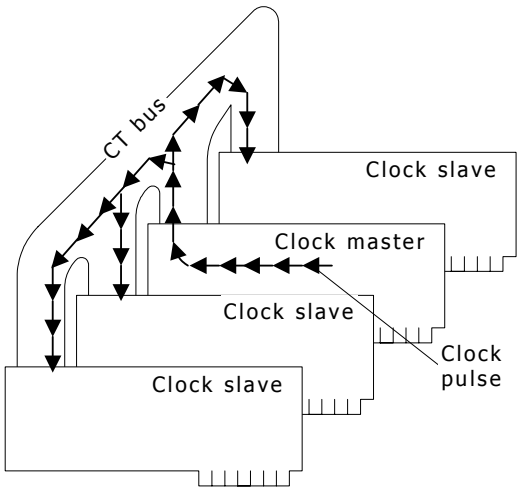
Note: Hardware clocking procedures are not transparent to the application. In addition to configuring clocking, the application must monitor clocking and take appropriate action when required.

Clock masters and clock slaves

To synchronize data transfer from device to device across the H.100 bus or H.110 bus, devices on the bus must be phase-locked to a high-quality 8 MHz clock and 8 kHz frame pulse. These signals together compose a CT bus clock.

One board on the bus generates (drives) the clock. This board is called the clock master. All other boards use this clock as a timing reference by which they synchronize their own internal clocks. These boards are called clock slaves (see the following illustration).

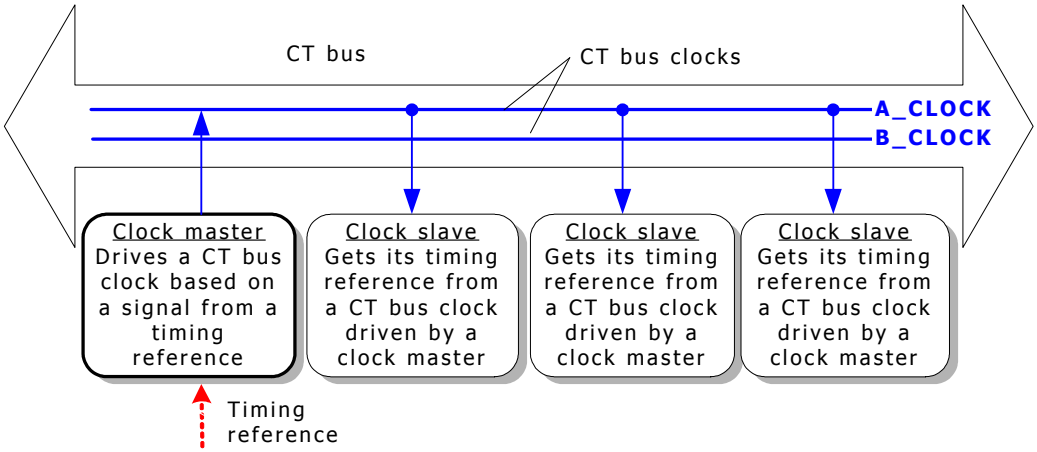
Note: Not all boards can serve as clock masters. For specifics, refer to your board documentation.



Clock master and clock slaves

Two CT bus clocks can run simultaneously on the bus. They are called A_CLOCK and B_CLOCK. The clock master can drive either one. When you set up CT bus clocking, choose one of these clocks for your master and slaves. The other one is a redundant signal that can be used by a secondary clock master (described in *Secondary clock masters* on page 92).

In the following illustration, the system is set up to use A_CLOCK:



System using A_CLOCK

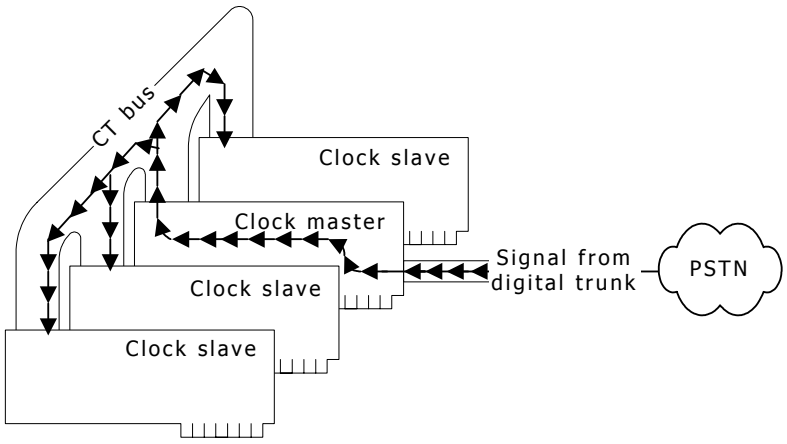
Timing references

To drive its CT bus clock, a clock master takes a reference signal, extracts the frequency information, defines a phase reference at the extracted frequency, and broadcasts this information as A_CLOCK or B_CLOCK. This reference signal is called a timing reference. When you set up a clock master, you specify what source the board uses as its timing reference.

Note: Not all boards support all timing references. For details on your board models, refer to the board documentation.

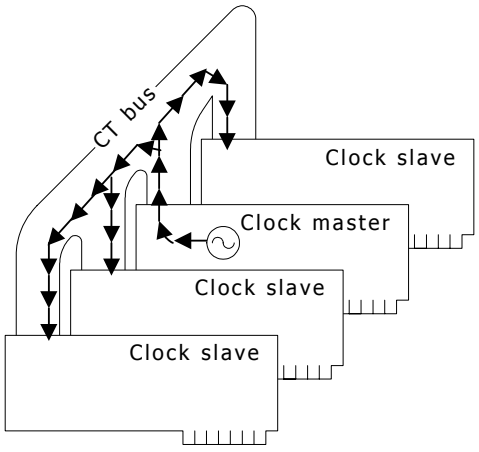
The timing reference signal originates in one of two places:

- It can originate within the public network and enter the system through a digital trunk. This is called a NETWORK timing reference.



Timing reference from NETWORK

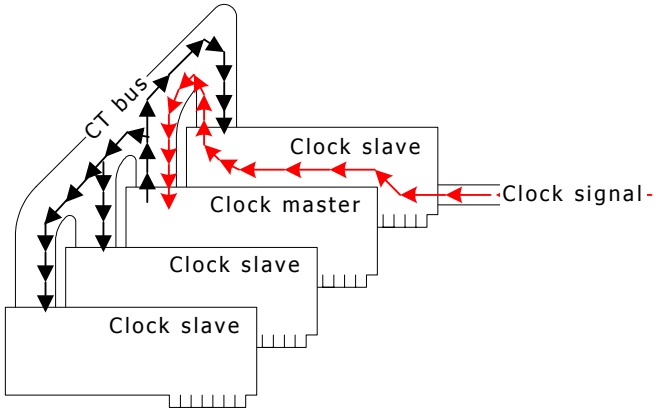
- In a system with no digital telephone network interfaces, an on-board oscillator can be used as the timing reference to drive the clock signals. This is called an OSC timing reference. Use OSC only if there is no external clock source available.



Timing reference from OSC

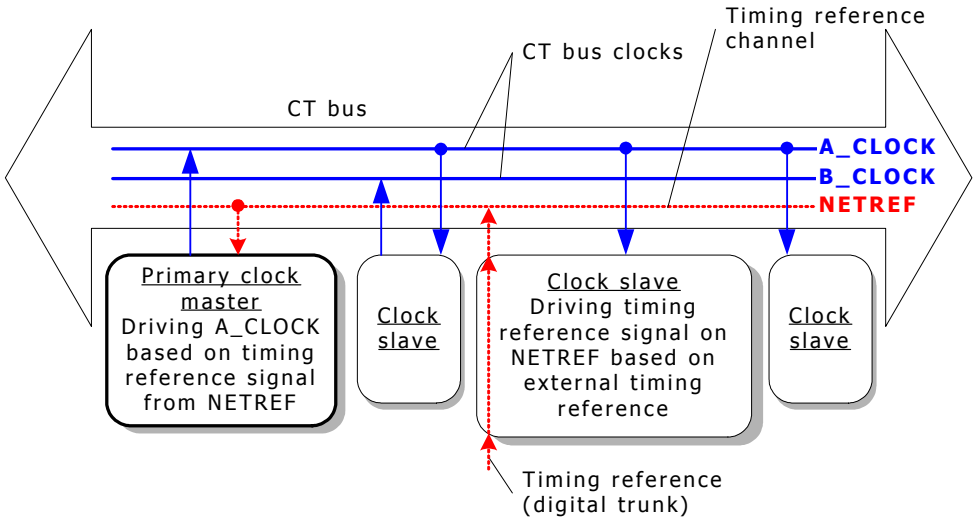
NETREF

The timing reference used by a clock master to drive the CT bus clock originates from an oscillator or trunk connected to another device in the system. In this case, the timing reference signal is carried over the CT bus to the clock master, which derives the clock signal and drives the clock for the slaves.



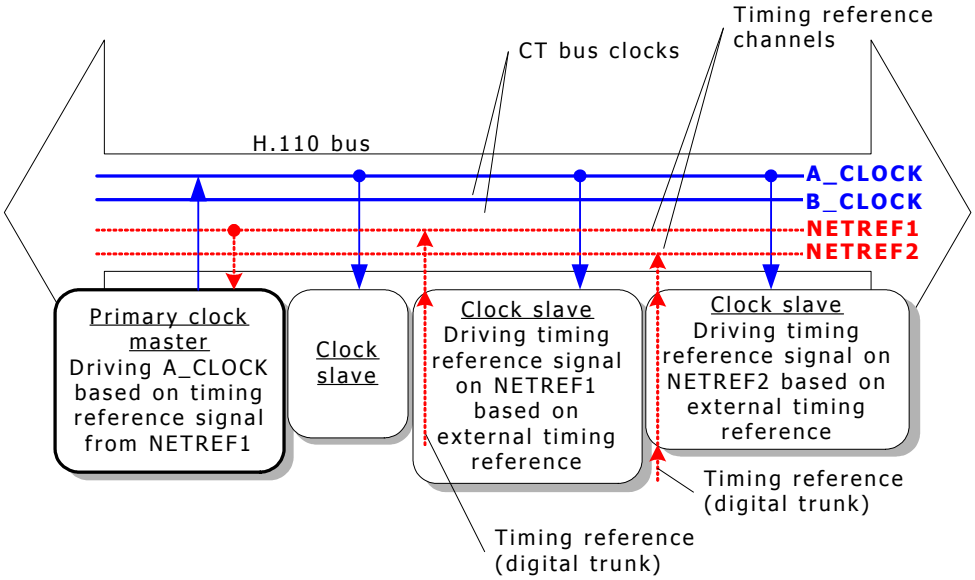
Timing reference from other device

The channel over which the timing reference signal is carried to the clock master is called NETREF.



NETREF

On the H.110 bus, a second timing reference signal can be carried on a fourth channel, called NETREF2. NETREF is referred to as NETREF1 in this case.



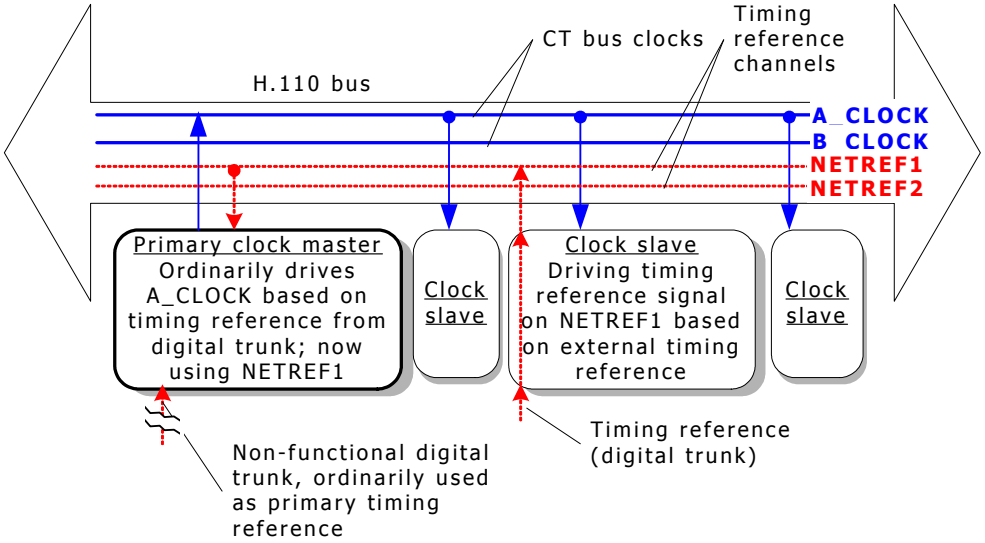
NETREF2

Note: Not all board models support NETREF or NETREF2. For more information about board models, refer to the board documentation.

Fallback timing references

Boards can optionally be assigned a backup (fallback) timing reference that it can use if its primary timing reference fails. For a clock master, the source for the fallback timing reference should NOT be the source currently used by the clock master for its primary timing reference.

For example, if a clock master's primary timing reference source is a NETWORK signal from one of its trunks, the fallback timing reference source can be a NETWORK signal from another one of its trunks, or a signal from NETREF1, NETREF2 (if H.110), or OSC. In the following illustration, the fallback timing reference source is NETREF1.



Fallback timing reference

The ability of a board to automatically switch to its fallback timing reference if its primary timing reference fails is called clock fallback. This feature can be enabled or disabled.

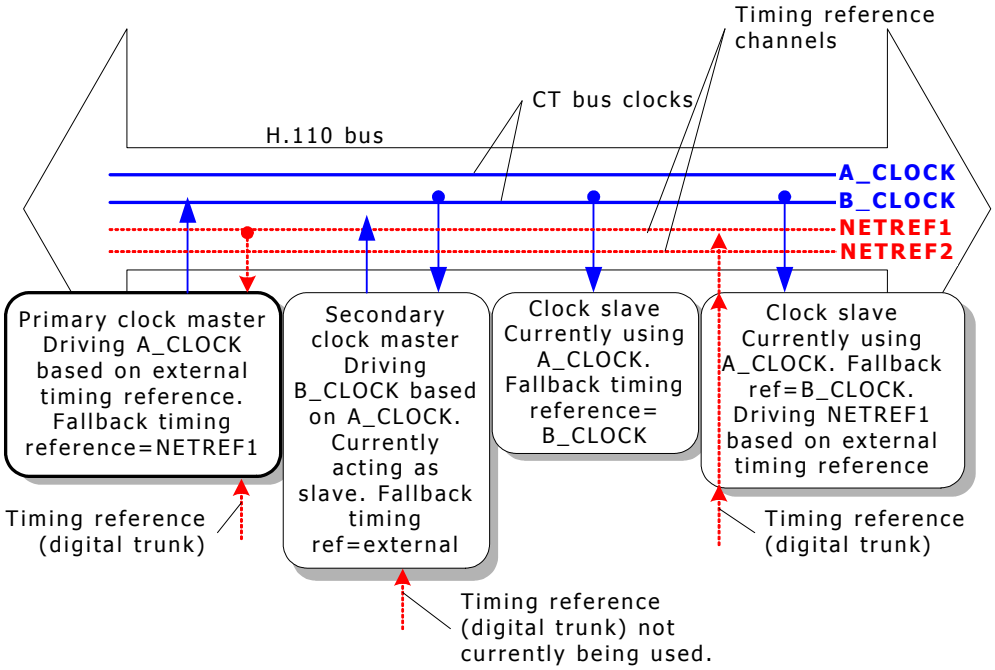
Note: Not all boards support clock fallback. For more information about board models, refer to the board documentation.

Secondary clock masters

You can set up a second device to be used as a backup, or secondary clock master, if the primary clock master stops driving its CT bus clock (because both of its timing references failed, or it was hot-swapped out, or for some other reason). For the secondary clock master to work:

1. It must receive its primary timing reference from the CT bus clock driven by the primary clock master (either A_CLOCK or B_CLOCK).
2. It must drive the CT bus clock not driven by the primary master. For example, if the primary clock master is driving A_CLOCK, the secondary clock master must drive B_CLOCK. In this case, both clocks are synchronized.
3. It must have a fallback timing reference. This timing reference must not be the primary clock master's primary or fallback timing reference.
4. All other slave boards must be set up so their fallback timing references are the CT bus clock driven by the secondary clock master.

The following illustration shows a sample secondary clock master configuration:



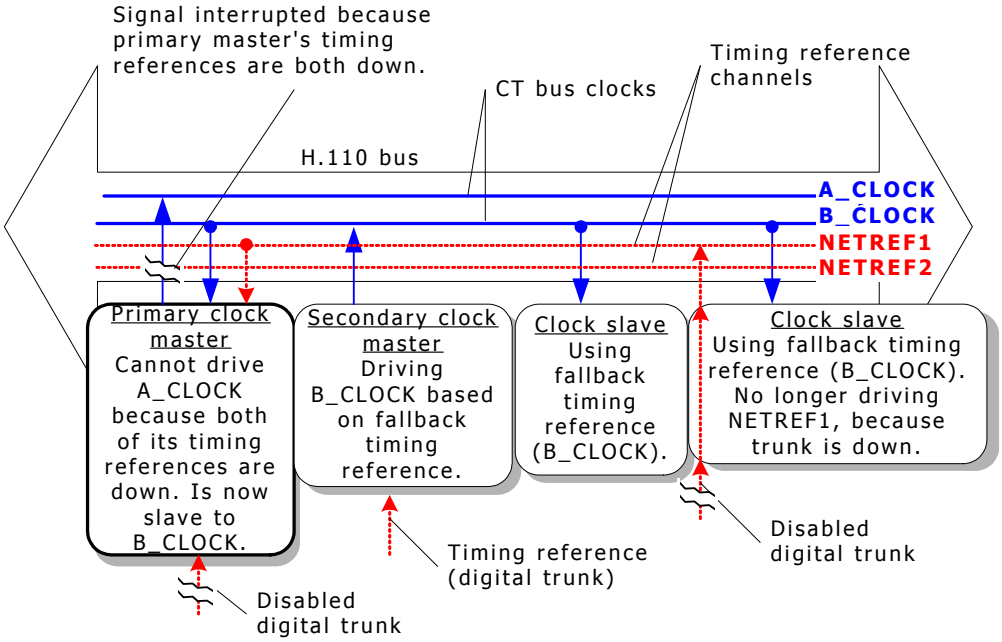
Fallback timing reference for secondary clock master

Note: Not all boards can act as secondary master. For details on your board models, refer to the board-specific documentation.

With a secondary clock master, clock fallback works as follows:

1. As long as the primary clock master is driving its CT bus clock, the secondary clock master acts as a slave to the primary clock master. However, the secondary master also drives the CT bus clock not driven by the primary master (for example, B_CLOCK if the primary master is driving A_CLOCK).
2. If the primary clock master stops driving its CT bus clock, all slaves (including the secondary clock master) lose their primary timing reference.
3. This triggers the secondary master to fall back to its fallback timing reference.
4. This also triggers other slaves to fall back to the CT bus clock driven by the secondary clock master.
5. The secondary master and slaves do not switch back to the primary timing reference automatically if the primary reference is re-established. Software intervention is required prior to any further clock changes.
6. If the board formerly used as the primary clock master is still active but is not receiving a primary or fallback timing reference, the board attempts to become a slave to the clock driven by the secondary master.

The secondary clock master is now clock master for the whole system, as shown in the following illustration:



Secondary clock master driving system

System configuration file example

The following example describes a system configuration where four boards reside in a single chassis. The boards are configured in the following way:

Board	Description	Drives	Primary timing reference	Fallback timing reference
A	Primary clock master	A_CLOCK	NETREF	Local digital trunk 2
B	Secondary clock master	B_CLOCK	A_CLOCK	Local digital trunk 3
C	Clock slave	Nothing	A_CLOCK	B_CLOCK
D	Clock slave	NETREF based on local digital trunk 4	A_CLOCK	B_CLOCK

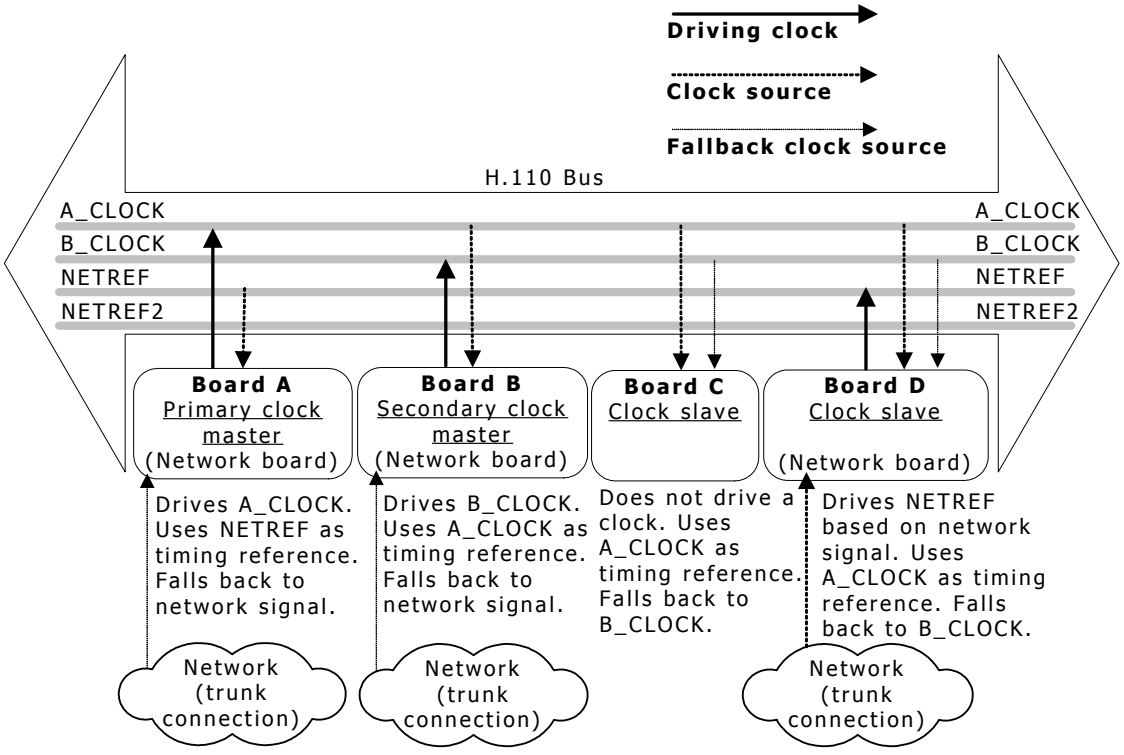
Clock fallback is enabled on all boards. Board A, defined as the primary clock master, drives A_CLOCK. All other boards on the system connected to the CT bus use A_CLOCK as their primary timing reference. Board A derives its own timing reference from the NETREF signal driven by board D, based on a signal from one of board D's digital trunks (trunk 4).

In addition, board A is configured to use timing signals received on one of its own digital trunks (trunk 2) as its fallback timing reference. If NETREF fails, board A continues to drive A_CLOCK based on its fallback timing reference.

Board B is set up as a backup, or secondary clock master, driving the CT bus clock not driven by the primary clock master. Board B normally receives its timing reference from A_CLOCK, which is driven by board A. This means that board B acts as a clock slave to board A. If A_CLOCK fails, board B continues driving B_CLOCK, but now uses the timing signals received from one of its digital trunks (trunk 3). All other slave boards fall back to B_CLOCK, and board B serves as the clock master. The primary master also falls back to B_CLOCK, and is now a slave to the secondary master. The system continues in this configuration until an application intervenes.

This configuration assigns the following clocking priorities:

Timing priority	Clocking configuration
First	Board A (primary master) drives A_CLOCK using its primary timing reference (board D, digital trunk 4, using NETREF). Slaves synchronize to A_CLOCK.
Second	Board A (primary master) drives A_CLOCK using its fallback timing reference (board A, digital trunk 2). Slaves synchronize to A_CLOCK.
Third	Board B (secondary master) drives B_CLOCK using its fallback timing reference (board B, digital trunk 3). Slaves synchronize to B_CLOCK.



Sample board clocking configuration

Board keywords

If clocking is set up for the system through board keywords, clock configuration keywords can be set as follows for each board:

Board	Role	Clocking keyword settings
A	Primary clock master	Clocking.HBus.ClockMode = MASTER_A Clocking.HBus.ClockSource = NETREF Clocking.HBus.AutoFallBack = YES Clocking.HBus.FallBackClockSource = NETWORK Clocking.HBus.FallBackNetwork = 2
B	Secondary clock master	Clocking.HBus.ClockMode = MASTER_B Clocking.HBus.ClockSource = A_CLOCK Clocking.HBus.AutoFallBack = YES Clocking.HBus.FallBackClockSource = NETWORK Clocking.HBus.FallBackNetwork = 3
C	Clock slave	Clocking.HBus.ClockMode = SLAVE Clocking.HBus.ClockSource = A_CLOCK Clocking.HBus.AutoFallBack = YES Clocking.HBus.FallBackClockSource = B_CLOCK
D	Slave driving NETREF	Clocking.HBus.ClockMode = SLAVE Clocking.HBus.ClockSource = A_CLOCK Clocking.HBus.AutoFallBack = YES Clocking.HBus.FallBackClockSource = B_CLOCK Clocking.HBus.NetRefSource = NETWORK Clocking.HBus.NetRefSourceNetwork = 4 Clocking.HBus.NetRefSpeed = 8K

Clock signal summary

The following table summarizes the reference clocks that a clock master can drive:

Clock	Details
A_CLOCK	The set of primary bit clocks (CT8A) and framing signals (CTFrameA). The CT8A signal is an 8 MHz clocking reference for transferring data over the CT bus. The CTFrameA provides a low going pulse signal every 1024 (8 MHz) clock cycles.
B_CLOCK	The set of secondary bit clocks (CT8B) and framing signals (CTFrameB). The CT8B signal is an 8 MHz clocking reference for transferring data over the CT bus. The CTFrameB provides a low going pulse signal every 1024 (8 MHz) clock cycles.

The following table summarizes the timing references that a clock master can use:

Timing reference	Details
NETWORK	The timing signal from a digital trunk attached to the clock master board. Within the digital trunk interface, an 8 kHz reference is derived from the frequency of the incoming signal. The clock master is frequency-locked to this 8 kHz reference so that the long-term timing of the system matches that of the public telephone network. **Note:** No timing signal is available from an analog trunk.
NETREF / NETREF1	The CTNETREF_1 signal. Can be 8 kHz, 1.544 MHz, or 8 MHz. NMS recommends using only 8 kHz signals for most boards.
NETREF2	(H.110 only) The CTNETREF_2 signal. Can be 8 kHz, 1.544 MHz, or 8 MHz. NMS recommends using only 8 kHz signals for most boards.
OSC	Clock signal derived from an oscillator on the clock master board. **Note:** Use this timing reference source only if no network timing references are available.

Note: Not all boards support all signals. For more information about board models, refer to the board documentation.

Configuring clocking in your system

Configure board clocking in your system in one of two ways. Choose only one of these configuration methods across all boards on the CT bus. Otherwise, the two methods interfere with one another, and board clocking does not operate properly.

Method	Details
Using *clockdemo* application model	Create an application that assigns each board its clocking mode, monitors clocking changes, and re-configures clocking if clock fallback occurs. A sample clocking application, *clockdemo*, is provided with Natural Access. *clockdemo* provides a robust fallback scheme that suits most system configurations. *clockdemo* source code is included, allowing you to modify the program if your clocking configuration is very complex. For more information, see *Running clockdemo* on page 99. **Note**: Most clocking applications (including *clockdemo*) require all boards on the CT bus to be started in standalone mode.
Using board keywords (with or without application intervention)	This method is documented in *Configuring the primary clock master* on page 112, *Configuring the secondary clock master* on page 113, and *Configuring clock slaves and standalone boards* on page 114. Unlike the *clockdemo* application, which allows several boards to take over mastery of the clock in a fallback situation, the board keyword method allows you to specify only a fixed primary and secondary master. For this reason, the board keyword method is best used only if you do not want to implement clock fallback in your system, or in test configurations where clock reliability is not a factor. The board keyword method does not create an autonomous clock timing environment. An application must still intervene when clock fallback occurs to reset system clocking before other clocking changes occur. If both the primary and secondary clock masters stop driving the clocks (and an application does not intervene), the boards default to standalone mode.

Choosing master and slave boards

Some boards can drive clock signals more reliably than others and have more flexible clocking capabilities. If your system contains several board models, choose the boards with the best clocking characteristics for your primary and secondary masters.

The following list ranks the NMS boards by their abilities to serve as clock masters:

1. CG 6***xxx*** family (best)
2. AG 4000, AG 4000C, AG 4040, and AG 4040C
4. AG 2000, AG 2000C, and AG 2000-BRI
5. CX 2000 and CX 2000C
6. QX 2000

For example, if your system contains a CG 6000C, an AG 4000C, and a CX 2000C board, the CG 6000C board should serve as primary master. The AG 4000C board should serve as secondary master. The CX 2000C board should act as a slave.

If you have more than one board of a given model, assign these boards as your masters before using any boards with poorer clocking characteristics. For example, if your system contains two AG 4000 boards and one AG 2000 board, the two AG 4000 boards should serve as primary and secondary masters. The AG 2000 board should act as a slave.

10 System-level clocking with clockdemo

Running clockdemo

clockdemo is an application that configures H.100 and H.110 clocks for all boards using switching commands. Then it monitors for clocking changes, and reconfigures clocking if clock fallback occurs or if the status of a timing reference changes.

clockdemo provides a robust fallback scheme that suits most system configurations. Source code is included, allowing you to modify the program if your clocking configuration is complex. For a complete overview of H.100 and H.110 bus clocking, refer to *CT bus clocking overview* on page 87.

Name

clockdemo

Purpose

- Configures H.100 and H.110 clocks for all boards using switching commands.
- Monitors for clocking changes and reconfigures clocking if clock fallback occurs or the status of a timing reference changes.

Usage

`clockdemo` [***options***]

where ***options*** are:

Option	Description
-b ***n***	Specifies a list of boards to be managed by *clockdemo*. Multiple boards can be specified by repeating the -b option on the command line: `-b 1 -b 2 -b 3` The minimum number of boards allowed is 2. In addition to the listed boards, clocking for all boards listed in the file specified with the -f option are managed. If no boards are specified using either method, *clockdemo* attempts to manage clocking for boards numbered 0, 1, and 2.
-f ***file_name***	Specifies the name of a text file containing a list of available timing references, prioritized by their reliability. For more information about this file, see *Creating a timing reference priorities file* on page 101. In addition to the boards listed in the file, clocking for all boards listed using the -b option are managed.
-d	Turns on debug messages, showing clocking status.
-t ***m_sec***	Specifies the rate (in ms) at which *clockdemo* polls the boards for clocking status changes. Default is 2000.
-h	Displays a Help screen and terminates.

Description

clockdemo configures H.110/H.100 clocking for all specified boards. It then monitors the boards (and board trunks), and reconfigures clocking if events occur that jeopardize or interrupt the clock pulse.

clockdemo is given a text file listing boards to manage. Also listed are digital trunks on these boards from which a clock master can derive a timing reference. In the file, the trunks are rated in order of reliability. Based on this file, *clockdemo* assigns the board with the most reliable available trunks as primary master, and configures the most reliable trunks on this board as the primary and fallback timing references for the system. *clockdemo* also attempts to configure a different board as secondary master. All other boards become clock slaves.

clockdemo then polls all boards at regular intervals, and monitors all trunks currently serving as timing references. If *clockdemo* detects that a system clock has failed, or that fallback has occurred, or that a critical trunk is having problems, it reassigns timing references or configures new primary and secondary masters to maintain or restore the integrity of the system clock. In each case, *clockdemo* picks the most reliable available boards and trunks.

clockdemo also monitors NMS OAM events to detect Hot Swap insertion or removal of boards. If the primary or secondary master is removed, *clockdemo* assigns a new primary or secondary master as necessary.

clockdemo logs clocking configurations and changes to the screen. The following is sample *clockdemo* output:

```
CLOCKDEMO Version 2.0 Mar 15 2001 15:00:36
Boards  Clock Mode   Current        Mastering         Fallback
                     Clock Source   A Clock B Clock   Occurred
-------------------------------------------------------------
  0:    SLAVE        H100 A                           NO
  3:    PRIMARY      NETWORK 1        YES             NO
  1:    SECONDARY    H100 A                   YES     NO
  2:    SLAVE        H100_A                           NO
-------------------------------------------------------------
 (Press 'q' then ENTER to Exit.)
```

clockdemo uses Switching (SWI) service commands to configure clocking on each board. It can configure and manage clocking on any board that complies with the MVIP-95 standard.

clockdemo can also operate without a clocking priorities file, and instead configure and manage clocking on boards specified on the command line with the -b option. In this case, *clockdemo* attempts to determine the current configuration and to maintain clock synchronization as best as it can. However, this mode does not provide as robust a clocking scheme as when the clocking priorities file is used. In some cases, *clockdemo* can assign OSC as a fallback source. Use this *clockdemo* mode only if board clocking has previously been configured using a *swish* script or other method.

If the clocking priorities file is used, set all boards to standalone mode before running *clockdemo*. To do so, use the NMS OAM utilities or the NMS OAM service to set the Clocking.HBus.ClockMode keyword to STANDALONE for each board.

Note: When using *clockdemo*, do not use any other Clocking.HBus.***XXX*** keywords to specify clocking configurations for your boards. Board-level clocking configuration interferes with *clockdemo*'s operation and can cause glitches in the system clock signal.

clockdemo does not provide support for NETREF(1) or NETREF2.

Creating a timing reference priorities file

The timing reference priorities file is specified with the -f option on the command line. It lists boards and trunks on the boards, and rates their reliability. *clockdemo* uses this information to determine how best to configure the boards, and which boards or timing references to use in case of signal failure.

The timing reference priorities file is an ASCII text file. In the file, trunks are listed one to a line, in this fashion:

priority board_number trunk_number

where:

Parameter	Description
priority	Indicates the reliability of the trunk. ***priority*** is an integer between 0 (best) and 99 (worst). Trunks with equivalent reliability can be given identical priority numbers.
board_number	Indicates the number of the board on which the trunk is located. ***board_number*** is an integer between 0 and 32767.
trunk_number	Indicates the trunk number. ***trunk_number*** is an integer between 0 and the total number of trunks supported by the board type. 0 designates the board's internal oscillator (OSC).

The values on each line are separated by spaces. Any text following a number sign (#) denotes a comment and is ignored.

clockdemo follows these rules when choosing primary and secondary masters and timing references for each:

- The system must have one (and only one) primary master.
- If possible, the system should also have one (and only one) secondary master.
- The primary and secondary masters must be different boards.
- The primary and fallback timing references for a master board must be trunks on that board or the board's internal oscillator.
- In case of a need to reconfigure clocking, reconfigure as few boards as necessary to maintain system integrity.

Sample timing reference priorities file

The following is a sample listing of a timing reference priorities file:

```
# A list of timing references that are prioritized
#       Priority        Board      Trunk
         0                  0          1
         0                  0          2
         0                  3          1
         1                  1          1
         99                 0          0      # OSC
         99                 1          0      # OSC
         99                 2          0      # OSC
         99                 3          0      # OSC
# end of list
```

The text in this file denotes the following:

- Board 0, trunks 1 and 2, and board 3, trunk 1 are all equally reliable, and are the most reliable trunks available.
- Board 1, trunk 1 is also available as a timing reference, but is not as reliable as the trunks listed above it.
- Boards 0, 1, 2, and 3 also have internal oscillators that can be used as timing references, but only as a last resort.

Assuming all trunks are non-operational to begin with, *clockdemo* makes the following initial assignments based on this file:

- Board 0 is primary master, driving A_CLOCK using its internal oscillator as a timing reference.
- Board 1 is secondary master, driving B_CLOCK based on A_CLOCK.
- Boards 2 and 3 are slaves to A_CLOCK.

clockdemo displays the configuration as follows:

```
CLOCKDEMO Version 2.0 Mar 15 2001 15:00:36
Boards   Clock Mode   Current        Mastering         Fallback
                      Clock Source   A Clock B Clock   Occurred
-------------------------------------------------------------------
  0:     PRIMARY      INTERNAL         YES               NO
  3:     SLAVE        H100 A                             NO
  1:     SECONDARY    H100 A                    YES      NO
  2:     SLAVE        H100_A                             NO
-------------------------------------------------------------------
 (Press 'q' then ENTER to Exit.)
```

Board 0 is primary master, driving A_CLOCK using its internal oscillator. Board 1 is secondary master. Its fallback timing reference is OSC. All other boards are clock slaves.

If board 1, trunk 1 becomes operational, the clock configuration changes as follows:

```
Boards   Clock Mode   Current        Mastering         Fallback
                      Clock Source   A Clock B Clock   Occurred
-------------------------------------------------------------------
  0:     SECONDARY    H100 A                    YES      NO
  3:     SLAVE        H100_A                             NO
  1:     PRIMARY      NETWORK 1        YES               NO
  2:     SLAVE        H100 A                             NO
-------------------------------------------------------------------
 (Press 'q' then ENTER to Exit.)
```

Board 1 has become primary master, driving A_CLOCK using trunk 1 as its primary timing reference. (Its secondary timing reference is OSC.) Board 0 has been demoted to secondary master. (Its secondary timing reference is OSC.)

If board 3, trunk 1 then becomes operational, board 3 becomes primary master, since it has a higher priority rating in the timing reference priorities file. (Its fallback timing reference is OSC.) Board 1 is demoted to secondary master. (Its fallback timing reference is trunk 1.) Board 0 is demoted to slave:

```
Boards   Clock Mode   Current        Mastering         Fallback
                      Clock Source   A Clock B Clock   Occurred
-------------------------------------------------------------
   0:    SLAVE        H100 A                           NO
   3:    PRIMARY      NETWORK 1        YES             NO
   1:    SECONDARY    H100 A                   YES     NO
   2:    SLAVE        H100 A                           NO
-------------------------------------------------------------
 (Press 'q' then ENTER to Exit.)
```

Now board 0, trunk 2 becomes operational. Simultaneously, board 0, trunk 1 momentarily becomes operational, and then stops. *clockdemo* creates the following configuration:

```
Boards   Clock Mode   Current        Mastering         Fallback
                      Clock Source   A Clock B Clock   Occurred
-------------------------------------------------------------
   0:    SECONDARY    H100_A                   YES     NO
   3:    PRIMARY      NETWORK 1        YES             NO
   1:    SLAVE        H100 A                           NO
   2:    SLAVE        H100 A                           NO
-------------------------------------------------------------
 (Press 'q' then ENTER to Exit.)
```

Now board 0, trunk 2 becomes non-operational:

```
Boards   Clock Mode   Current        Mastering         Fallback
                      Clock Source   A Clock B Clock   Occurred
-------------------------------------------------------------
   0:    SLAVE        H100_A                           NO
   3:    PRIMARY      NETWORK 1        YES             NO
   1:    SECONDARY    H100 A                   YES     NO
   2:    SLAVE        H100 A                           NO
-------------------------------------------------------------
 (Press 'q' then ENTER to Exit.)
```

Now board 3, trunk 1 becomes non-operational:

```
Boards   Clock Mode   Current        Mastering         Fallback
                      Clock Source   A Clock B Clock   Occurred
-------------------------------------------------------------
   0:    SECONDARY    H100 A                   YES     NO
   3:    SLAVE        H100_A                           NO
   1:    PRIMARY      NETWORK 1        YES             NO
   2:    SLAVE        H100 A                           NO
-------------------------------------------------------------
 (Press 'q' then ENTER to Exit.)
```

Lastly, board 1, trunk 1 becomes non-operational:

```
Boards   Clock Mode   Current        Mastering         Fallback
                      Clock Source   A Clock B Clock   Occurred
-------------------------------------------------------------
   0:    SECONDARY    H100 A                   YES     NO
   3:    SLAVE        H100_A                           NO
   1:    PRIMARY      INTERNAL         YES             NO
   2:    SLAVE        H100 A                           NO
-------------------------------------------------------------
 (Press 'q' then ENTER to Exit.)
```

clockdemo program structure

clockdemo consists of the following files:

File	Contains
clockdemo.c	Main source code.
clockdemo.exe	Compiled executable.
clockresource.c	Auxiliary functions invoked by functions in *clockdemo* to manage timing reference information such as, trunks.
clockresource.h	Defines for *clockresource.c*.
priority.txt	Sample clocking priority list file.

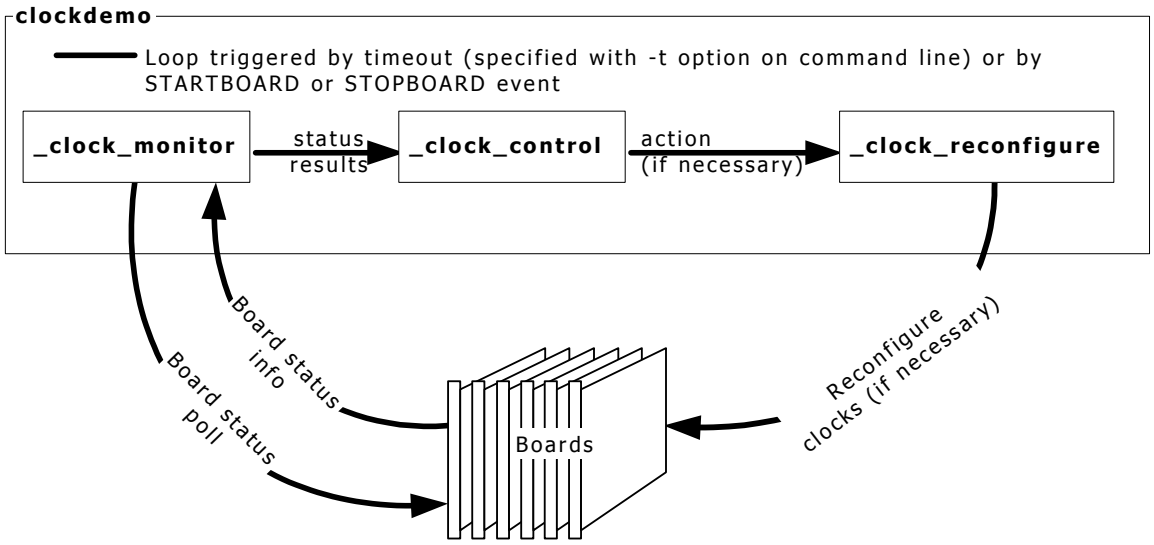
clockdemo performs three main tasks:

- Monitors for clocking changes (for example, trunk signal failures).
- Determines what to do when there is a change.
- Reconfigures clocking on the boards.

When a change to a timing reference or board requires changes to the clocking configuration, *clockdemo* determines the minimum number of steps to take to rectify the situation, to avoid reconfiguring boards unnecessarily.

The code for each of *clockdemo*'s tasks is encapsulated in the main section of the program. The loop is triggered at regular intervals (regulated by the timeout set using the -t option on the command line). It is also triggered whenever an NMS OAM board started or board stopped event is received. The main section contains the following functions:

Function	Description
_clock_monitor	Polls all boards and returns the board status results in an array.
_clock_control	Determines an action to take (if necessary) for each board based upon the status results.
_clock_reconfigure	Reconfigures affected boards based upon the determined action.



clockdemo main functions

main

The main section of the program does the following:

1. Initializes variables and parses the command line options.
2. Invokes **_initData_file** to read and parse the configuration file.
3. Attempts to open the switch for each listed board (by invoking **swiOpenSwitch**), and builds an array of switch handles (swihd_array[]).
4. Based on the information from the configuration file, determines the primary and fallback timing references for the primary master, and the fallback timing reference for the secondary master.
5. Builds an array (h110_parms[]) containing the new configuration for each affected board.
6. Configures the primary and secondary master by invoking **swiConfigBoardClock** with h110_parms[].
7. Registers to receive NMS OAM events.
8. Invokes **swiGetBoardClock** for each board to determine its clocking status, and stores the results in an array (h110_query_parms[]).
9. Invokes **_printClockStatus** to print the initial clocking status to the screen.
10. Enters the while loop described in the previous illustration.
11. Updates the screen based on clocking changes.
12. Shuts down if the Q key is pressed.

_clock_monitor

_clock_monitor is called at regular intervals by the while loop in the main section. Its purpose is to determine whether a clocking change has occurred by monitoring the clocking status of each board and the status of each trunk involved in clocking. It returns an integer indicating one or more of six possible states. This information is used by the **_clock_control** function to determine whether to perform any clocking reconfiguration.

_clock_monitor is passed the following information:

Parameter	Description
num_of_boards	The number of boards in the chassis
swihd_array[]	An array containing the handles of all boards opened in the main section of the program.

_clock_monitor returns the following information:

Parameter	Description
clock_status	An integer indicating the type of clocking change (if any): • 0: No clocking change has occurred. • CLKSYS_STATUS_FALLBACK_OCCURRED (0x01): Fallback has occurred. • CLKSYS_STATUS_A_FAIL (0x02): A_CLOCK is no longer being driven. • CLKSYS_STATUS_B_FAIL (0x04): B_CLOCK is no longer being driven. • CLKSYS_STATUS_INCONSISTENT (0x08): Fallback has occurred on some boards but not all boards. This can occur if the primary master is switching between its primary and fallback timing references, but is still managing to drive the system clock successfully. • CLKSYS_STATUS_REFERENCE_CHANGED (0x10): A_CLOCK and B_CLOCK have not failed, but one of the trunks designated as a fallback source has failed.
h110_query_parms[]	An updated array containing the clocking status information for each board. This information consists of the following: • Its A_CLOCK status • Its B_CLOCK status • Whether fallback occurred or not
(Managed in *clockresource.c*)	A global structure containing the updated status of each trunk on each board.

_clock_monitor does the following:

1. Checks each board in swihd_array[] to determine whether it exists.
2. Polls the status of each existing board using **swiGetBoardClock**. This updates h110_query_parms[] for the board.
3. If more than 50 percent of the boards report that A_CLOCK failed, includes CLKSYS_STATUS_A_FAIL in the returned clock_status.
4. If more than 50 percent of the boards report that B_CLOCK failed, includes CLKSYS_STATUS_B_FAIL in the returned clock_status.
5. If fallback is reported by a board, includes CLKSYS_STATUS_FALLBACK_OCCURRED in the returned clock_status.
6. If fallback is reported by a board, but there is no clock failure, includes CLKSYS_STATUS_INCONSISTENT in the returned clock_status.
7. If one of the trunks designated as a fallback source has failed, includes CLKSYS_STATUS_REFERENCE_CHANGED in the returned clock_status.
8. Updates the global structure containing the updated status of each trunk on each board.

_clock_control

_clock_control is called at regular intervals by the while loop in the main section, directly after **_clock_monitor** returns. It uses the information returned by **_clock_monitor** to determine if any action is necessary to reconfigure the clocks. It returns an action code describing the action to be taken (if any).

_clock_control is passed the following information:

Parameter	Description
clock_status	Code returned by **_clock_monitor** describing the status of the system clocks and timing references.
h110_parms[]	The previous set of clocking settings made to each board by *clockdemo*.
h110_query_parms[]	Array containing the clocking status information for each board, updated by **_clock_monitor**.
num_of_boards	Number of boards in the system.

_clock_control returns the following information:

Parameter	Description
action	An integer describing one or more actions to take (if any): • CLKSYS_ACTION_NONE (0x0): Take no action. • CLKSYS_ACTION_RELOAD (0x01): Reload each board's current configuration. • CLKSYS_ACTION_NEW_PRIMARY (0x02): Configure a new primary master. • CLKSYS_ACTION_NEW_SECONDARY (0x04): Configure a new secondary master. • CLKSYS_ACTION_RELOAD_PRIMARY_FALLBACK (0x08): Reload the primary master's current configuration. Do not change any other boards.
h110_parms[]	An updated set of clock settings to make to one or more boards.

_clock_control does the following:

1. Calls **_find_master**. This function examines h110_parms[] and returns indices to the boards in the array configured as primary and secondary master. **_find_master** also determines which clock (A_CLOCK or B_CLOCK) is being used as the primary system clock.
2. If the primary clock (A_CLOCK or B_CLOCK) has failed:
    - Calls **_choose_new_primary** to choose a new primary master based on the configuration file.
    - Calls **_choose_new_secondary** to choose a new secondary master based on the configuration file.
    - Updates the h110_parms[] array describing the new system-wide configuration.
    - Includes CLKSYS_ACTION_NEW_PRIMARY and CLKSYS_ACTION_NEW_SECONDARY in the returned action.
3. If the primary clock (A_CLOCK or B_CLOCK) is functional, but the secondary clock (B_CLOCK or A_CLOCK) failed:
    - Calls **_choose_new_secondary** to choose a new secondary master based on the configuration file.
    - Updates the h110_parms[] array to describe a new configuration in which the old secondary master is now a slave.
    - Includes CLKSYS_ACTION_NEW_SECONDARY in the returned action.
4. If CLKSYS_STATUS_INCONSISTENT was returned by **_clock_monitor**:
    - Updates the h110_parms[] array to describe a configuration in which all secondary master and slave boards reporting clock inconsistency are reloaded with their current configurations.
    - If the primary master reported an inconsistency, selects a new fallback source for the primary master.
    - Includes CLKSYS_ACTION_RELOAD in the returned action.
5. If CLKSYS_STATUS_REFERENCE_CHANGED was returned by **_clock_monitor**:
    - Determines new fallback sources based on the priorities set in the configuration file.
    - If the source is on another board, chooses a new secondary master.
    - Updates h110_parms[] to describe the new configuration.
    - Includes CLKSYS_ACTION_NEW_PRIMARY, CLKSYS_ACTION_NEW_SECONDARY, and/or CLKSYS_ACTION_RELOAD_PRIMARY_FALLBACK in the returned action as appropriate.

_clock_reconfigure

_clock_reconfigure is called at regular intervals by the while loop in the main section. It is called directly after **_clock_control** returns, unless action is CLKSYS_ACTION_NONE. It reconfigures one or more of the boards in the system based upon the action returned by **_clock_control**.

_clock_reconfigure is passed the following information:

Parameter	Description
action	Action code returned by **_clock_control**.
h110_parms[]	Array updated by **_clock_control** describing the new configuration to set.
swihd_array[]	Array of switch handles.
num_of_boards	Number of boards in the system.
h110_query_parms[]	Array updated by **_clock_monitor** containing the current configuration as reported by **swiGetBoardClock**.

_clock_reconfigure does the following:

1. If action is CLKSYS_ACTION_RELOAD:
    - Compares various parameters in h110_query_parms[] (the configuration reported by the boards) with equivalent parameters in h110_parms[] (the new configuration specified by **_clock_control**).
    - If any of the parameters do not match, reconfigures the boards as specified in h110_parms[].
2. If action is CLKSYS_ACTION_NEW_PRIMARY and the board number of the primary master has changed:
    - Sets all boards to standalone mode.
    - Configures the primary master as specified in h110_parms[].
    - Configures the secondary master as specified in h110_parms[].
    - Configures the slaves as specified in h110_parms[].
3. If action is CLKSYS_ACTION_NEW_PRIMARY and the board number of the primary master has NOT changed:
    - Configures the primary master as specified in h110_parms[].
4. If action is CLKSYS_ACTION_NEW_SECONDARY:
    - Reconfigures the old secondary master as a slave, as specified in h110_parms[].
    - Configures the new secondary master as specified in h110_parms[].
5. If action is CLKSYS_ACTION_RELOAD_PRIMARY_FALLBACK:
    - Gets the primary master's old configuration, and reloads it. Does not disturb other boards.

11 Configuring CT bus clocking with board keywords

Limitations of clock configuration with board keywords

You can configure clocking in a system by specifying each board's role in the board's record in the NMS OAM database using keywords. Configuring clocking in this manner is best if you do not want to implement clock fallback in your system, or in test configurations where clock reliability is not a factor. Otherwise, for maximum system reliability, control clocking at the system level using an application such as *clockdemo.* For more information, refer to *Running clockdemo* on page 99.

For a detailed overview of H.100 and H.110 bus clocking, refer to *CT bus clocking overview* on page 87. For more information about retrieving and setting NMS OAM keyword values, refer to the *NMS OAM Service Developer's Reference Manual*.

Configuring clocking with board keywords has the following limitations:

- Unlike the *clockdemo* application, which allows you to specify several boards to take over mastery of the clock from one another in a fallback situation, the board keyword method allows you to specify only a fixed primary and secondary master.
- The board keyword method does not create an autonomous clock timing environment. If you implement clock fallback using this method, an application must still intervene when clock fallback occurs to reset system clocking before other clocking changes occur. If both the primary and secondary clock masters stop driving the clocks (and an application does not intervene), the boards default to standalone mode.

  Note: Not all boards can act as primary or secondary master. For information about board models, refer to the board documentation.

Refer to the *System configuration file example* on page 94 for an example of using board keywords to set up clock fallback in a multiple-board system.

Configuring the primary clock master

The primary clock master drives a CT bus clock used as the primary timing reference by all other boards connected to the CT bus. Use the following keywords to configure the primary clock master:

Keyword	Description
Clocking.HBus.ClockMode	Specifies the CT bus clock that the board drives. For the primary clock master, specify: • MASTER_A for A_CLOCK • MASTER_B for B_CLOCK
Clocking.HBus.ClockSource	Specifies the primary timing reference for the board. For the primary clock master, set to any of the following: • NETREF to use NETREF (also called NETREF1) • NETREF2 to use NETREF2 (H.110 only) • NETWORK to derive the timing from the clock pulse on a digital trunk connected to the board • OSC to use the board's on-board oscillator. Use only when no other source is available.
Clocking.HBus.ClockSourceNetwork	If Clocking.HBus.ClockSource is set to NETWORK, specifies the board trunk to derive the primary timing reference from (1 to ***n***, where ***n*** is the number of trunks on the board).
Clocking.HBus.AutoFallBack	Enables or disables clock fallback on the board. When set to YES, specifies that the board automatically switches to the Clocking.HBus.FallBackClockSource timing reference when the Clocking.HBus.ClockSource timing reference fails. The board continues to drive the CT bus clock using this timing reference until the first timing reference is re-established.
Clocking.HBus.FallBackClockSource	Specifies the fallback timing reference for the board to use if the primary timing reference fails. The board continues to drive the CT bus clock using this timing reference until the primary timing reference is re-established. For the primary clock master, set to any of the following: • NETREF to use NETREF (also called NETREF1) • NETREF2 to use NETREF2 (H.110 only) • NETWORK to derive the timing from the clock pulse on a digital trunk connected to the board • OSC to use the board's on-board oscillator. Use only when no other source is available. **Note**: The fallback timing reference must be different from the primary timing reference.
Clocking.HBus.FallBackNetwork	If Clocking.HBus.FallBackClockSource is set to NETWORK, specifies the board trunk from which to derive the fallback timing reference. (1 to ***n***, where ***n*** is the number of trunks on the board).
Clocking.HBus.NetRefSpeed	If using NETREF(1) or NETREF2 as a timing reference, set to 8K.

Configuring the secondary clock master

You can optionally set up a secondary clock master to drive a CT bus clock if the primary clock master stops driving its CT bus clock. Use the following keywords to configure the secondary clock master:

Keyword	Description
Clocking.HBus.ClockMode	Specifies the CT bus clock that the board drives. For the secondary clock master, specify the clock not driven by the primary clock master. For example, if the primary master drives B_CLOCK, specify MASTER_A for this keyword for the secondary master.
Clocking.HBus.ClockSource	Specifies the primary timing reference for the board. For the secondary clock master, set to the CT bus clock driven by the primary master: A_CLOCK or B_CLOCK. This makes the secondary master a slave to the primary master.
Clocking.HBus.AutoFallBack	Enables or disables clock fallback on the board. For the secondary clock master, set to YES.
Clocking.HBus.FallBackClockSource	Specifies the fallback timing reference for the board to use if the primary timing reference fails. Once the secondary master is driving the CT bus clock, it continues to drive the clock until software intervention by an application. For the secondary clock master, set to any timing reference not used by the primary clock master: • NETREF to use NETREF (also called NETREF1) • NETREF2 to use NETREF2 (H.110 only) • NETWORK to derive the timing from the clock pulse on a digital trunk connected to the board • OSC to use the board's on-board oscillator. Use only when no other source is available.
Clocking.HBus.FallBackNetwork	If Clocking.HBus.FallBackClockSource is set to NETWORK, specifies the board trunk from which to derive the fallback timing reference.
Clocking.HBus.NetRefSpeed	If using NETREF(1) or NETREF2 as a timing reference, set to 8K.

Configuring clock slaves and standalone boards

Any board connected to the CT bus that is not the primary or secondary clock master must be configured as a clock slave. Each clock slave derives its primary timing reference from A_CLOCK or B_CLOCK (whichever is driven by the primary clock master).

If you have set up a secondary clock master, when the primary clock master stops driving its CT bus clock, the clock slaves can get their clocking information from the secondary clock master.

Use the following keywords to configure clock slaves:

Keyword	Description
Clocking.HBus.ClockMode	Specifies the CT bus clock that the board drives. For a clock slave, set to SLAVE to indicate that the board does not drive any CT bus clock.
Clocking.HBus.ClockSource	Specifies the primary timing reference for the board. For each slave, set to the CT bus clock driven by the primary master: A_CLOCK or B_CLOCK.
Clocking.HBus.AutoFallBack	Enables or disables clock fallback on the board. If you have set up a secondary clock master, set to YES for each slave. Otherwise, set to NO.
Clocking.HBus.FallBackClockSource	Specifies the fallback timing reference for the board, to use if the primary timing reference fails. If you have set up a secondary clock master, set to the timing reference driven by the secondary clock master. Once a slave switches to the secondary clock, it continues to use the clock until reset by an application.

Configuring standalone boards

Configure a board in standalone mode so that the board references its own timing information, set Clocking.HBus.ClockMode to STANDALONE.

In this mode, the board is not able to make connections to the CT bus.

With some board models, specifying standalone mode can cause certain default switch connections to be made on the board to route incoming information from the trunk to DSP resources. The SwitchConnections and SwitchConnectMode keywords control this behavior. For details, see the board documentation.

Board-level clock fallback behavior

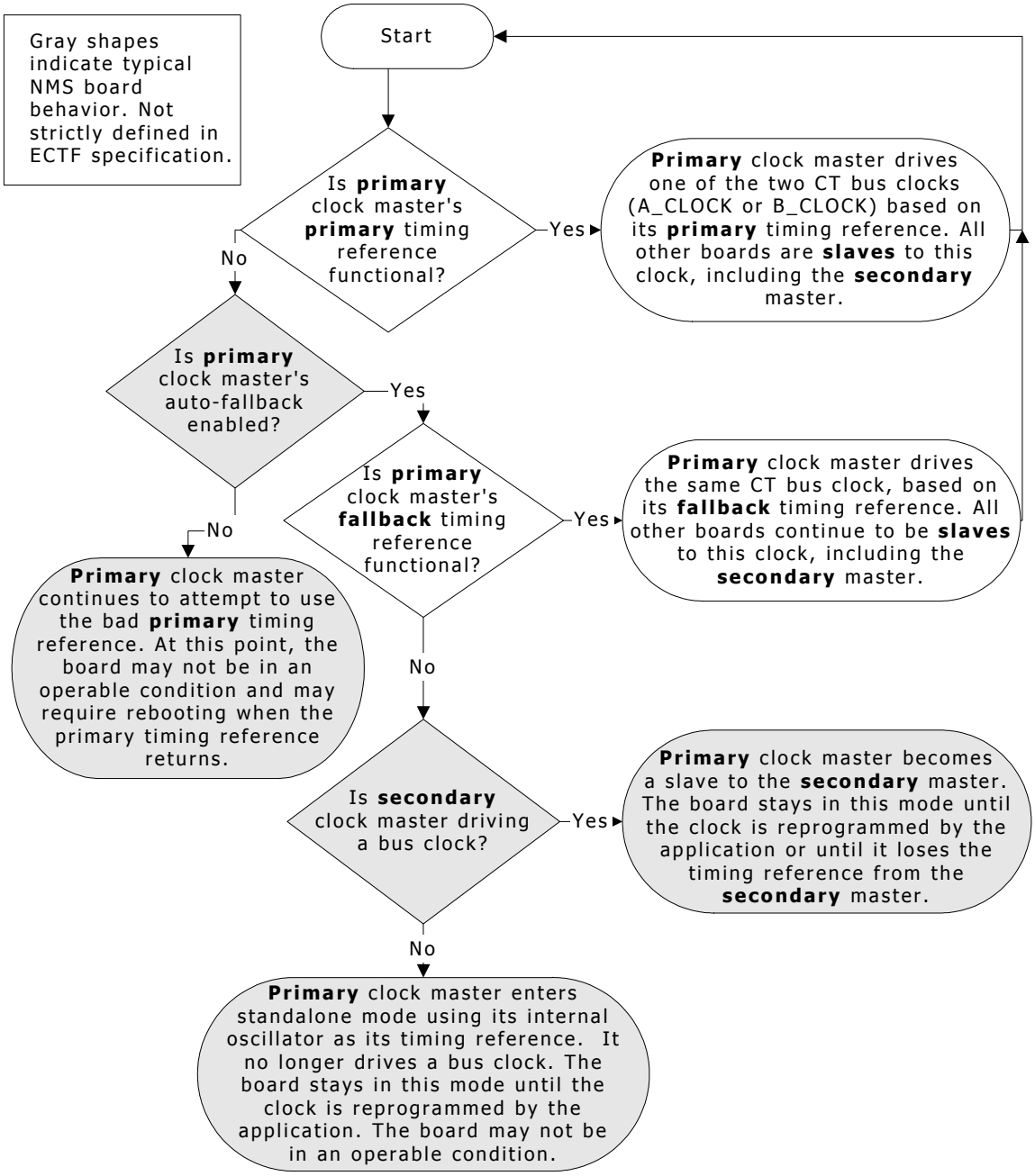
This topic describes the following aspects of clock fallback behavior when clocking is configured with board keywords:

- Primary clock master fallback behavior
- Secondary clock master fallback behavior
- Clock slave fallback behavior

Note: The illustrations describe the actions taken by most NMS board models in these situations. For specifics on a particular board, refer to the board manual.

Primary clock master fallback behavior

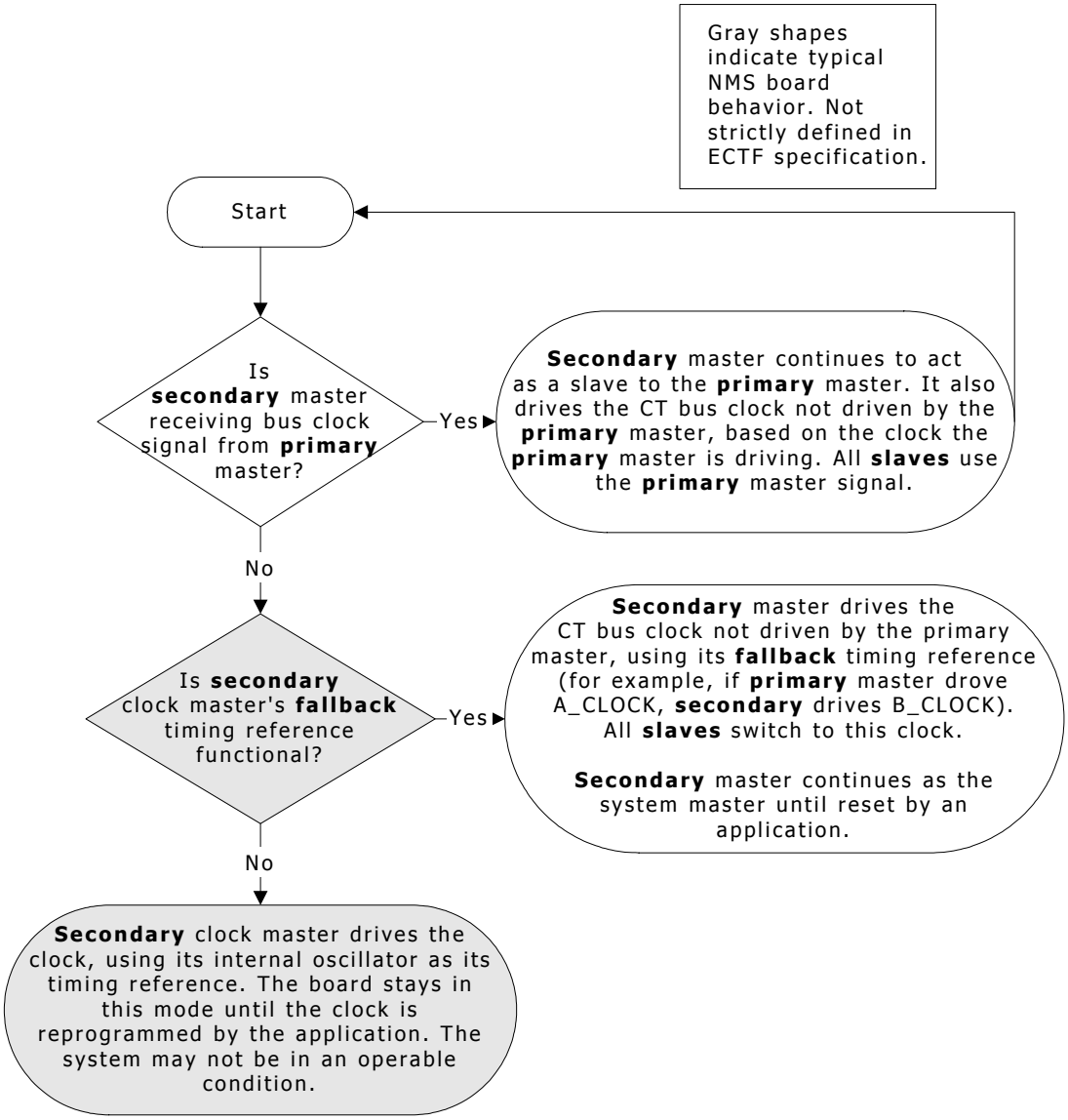
The following illustration shows the role of the primary clock master in board-level clock fallback. If the primary master loses its primary timing reference and switches to its secondary reference, and then the primary reference is established again, the master switches back to the primary timing reference.



Clock fallback behavior (primary clock master)

Secondary clock master fallback behavior

The following illustration shows the role of the secondary clock master in board-level clock fallback. The secondary master takes over when the primary master stops driving the clock. The secondary master continues to drive the clock for the system until an application intervenes.



Clock fallback behavior (secondary clock master)

Clock slave fallback behavior

If the primary master stops driving the clock, all slaves attempt to switch over to the other CT bus clock, driven by the secondary master. They continue to use this clock until reset by an application.

If fallback is enabled, but the secondary timing reference is not functional, the board enters standalone mode, using its internal oscillator as the timing reference. It continues in this fashion until the secondary timing reference is restored. The board continues using either the secondary timing reference or the oscillator until reset by an application.

For more information, refer to the illustration in Primary clock master fallback behavior.

Configuring NETREF (NETREF1) and NETREF2

If you specified that any board use NETREF (NETREF1) or NETREF2 as a timing reference, you must configure one or two other boards to drive the signals. Configure a different board for each signal. The source for each signal can be a digital trunk.

Note: NETREF2 is available only in H.110 configurations.

Use the following keywords to configure a board to drive NETREF (NETREF1):

Keyword	Description
Clocking.HBus.NetRefSource	Specifies the source of the NETREF (NETREF1) timing reference. Set to any of the following: • NETWORK to drive NETREF based on the signal from a digital trunk connected to the board. • STANDALONE if the board will not drive NETREF. • OSC to drive NETREF using the board's oscillator for debugging purposes only.
Clocking.HBus.NetRefSourceNetwork	If Clocking.HBus.NetRefSource is set to NETWORK, specifies the number of the trunk from which to get the signal.
Clocking.HBus.NetRefSpeed	Sets the speed of the NETREF signal. Set to 8K.

Use the following keywords to configure a board to drive NETREF2:

Keyword	Description
Clocking.HBus.NetRef2Source	Specifies the source of the NETREF2 timing reference. Set to any of the following: • OSC to drive NETREF2 using the board's oscillator. • NETWORK to drive NETREF2 based on the signal from a digital trunk connected to the board. • STANDALONE if the board will not drive NETREF2.
Clocking.HBus.NetRef2SourceNetwork	If Clocking.HBus.NetRefSource is set to NETWORK, specifies the number of the trunk from which to get the signal.
Clocking.HBus.NetRef2Speed	Sets the speed of the NETREF signal. Set to 8K.

Note: Not all boards can drive NETREF or NETREF2. For more information, refer to the board documentation.

12 Migrating to NMS OAM

Summary of changes

This section provides information on migrating from *agmon* to NMS OAM. The following list summarizes the changes introduced with NMS OAM:

- The AG board configuration and monitoring utility, *agmon*, is deprecated. NMS OAM performs board management operations across AG, QX, CX, and CG boards. Utilities included with NMS OAM duplicate and enhance operations formerly performed by *agmon*.
- The AGM library is deprecated. NMS OAM has an API for initializing and monitoring boards and for performing many other tasks.
- The AG configuration file has been replaced by files with very different structure and syntax. Keywords used in these files are different from AG configuration file keywords.

  The *ag2oam* utility, included with NMS OAM, translates AG configuration files into the new syntax.
- Previously, the only method of identifying a board in software was the board number. A new identifier, the name, can be used to identify each board, as well as certain software modules and other components.
- The HSI service is deprecated. Hot Swap functionality is implemented as an extended component of NMS OAM. The Hot Swap manager has not changed.
- The QX board configuration and monitoring utility, *qxload*, is deprecated. NMS OAM performs board management operations for QX boards. For information about migrating QX applications to NMS OAM, refer to the *QX 2000 Installation and Developer's Manual*.
- NMS SNMP services use NMS OAM. Therefore, SNMP provides information only on boards started using the OAM service.
- Tracing messages are retrieved using the NMS OAM *oammon* monitoring utility.

NMS OAM and agmon differences

agmon is deprecated with the introduction of NMS OAM. NMS OAM provides all functionality formerly provided by *agmon*. NMS OAM differs from *agmon* in the following ways:

- *agmon* is a utility program, controllable only using its command line. NMS OAM is a Natural Access service, accessible programmatically using its API. Various subsets of NMS OAM service functionality can also be accessed with the *oamsys*, *oamcfg*, and *oammon* utilities.
- *agmon* configures, boots, and monitors boards as a single operation. With NMS OAM, configuration, board starting and stopping, and monitoring operations are all accessible separately, using the NMS OAM utilities and API functions.
- With *agmon*, the central repository of configuration information is the AG configuration file. With NMS OAM, configuration information is kept in a dynamic database that is managed by the service.

  The utilities supplied with NMS OAM use configuration files as a convenient way to supply information to the NMS OAM database. However, when NMS OAM starts boards, the information in the database for each board (not the configuration files) determines how the board is configured.
- NMS OAM provides functionality not available with *agmon*, such as board testing (for board models that support this operation) and alert notification. It is also extensible with extended management components (EMCs) and board plug-ins. For example, Hot Swap is now implemented as an EMC.
- NMS OAM supports new board families such as CG.

Note: NMS OAM and *agmon* cannot be used simultaneously.

NMS OAM service utilities

The following utilities are supplied with NMS OAM:

Utility	Description
oamsys	Replaces *agmon*'s configuration and booting capabilities. Configures the NMS OAM database based on information supplied in configuration files and then starts all boards.
oamcfg	Provides access to individual NMS OAM configuration functions. Reads configuration files to configure individual boards.
oammon	Replaces *agmon*'s monitoring capabilities. Monitors boards for board errors and events.
oaminfo	Enables you to display and set NMS OAM keywords. Searches for text in keywords. For more information about *oaminfo*, refer to the *NMS OAM Service Developer's Reference Manual*.

Migrating configuration files

With *agmon*, all information for all boards was specified in a single AG configuration file. With NMS OAM utilities, a system configuration file contains a list of managed components in the system (boards or software modules, such as an EMC). For each managed component, a list is specified of parameters and values to configure that component. Most of the parameters for boards are usually listed in separate keyword files referenced in the system configuration file.

The syntax of these files is very different from the syntax of an AG configuration file. Parameters are still specified as keyword name and value pairs (for example, AutoStart = YES). However, struct keywords (containing multiple values) and array keywords (containing multiple indexed values) are now supported. These keywords are often specified using a special shorthand notation.

Keyword names are as consistent as possible across board families.

For more information about system configuration files, see *Creating a system configuration file* on page 43. For more information about keyword files, see *Using board keyword files* on page 46. For more information about NMS OAM equivalents for specific AG configuration file keywords, refer to the board documentation.

ag2oam

The *ag2oam* utility included with NMS OAM translates AG configuration files into system configuration files and keyword files that *oamsys* can process. To use *ag2oam*:

1. Go through the AG configuration file and determine the product type for each board number. For example, Board 0 = AG 2000 T1; Board 1 = AG 2000 T1; Board 2 = AG 4000C T1.
2. Enter:

   ```
   ag2oam [options]
   ```

   where ***options*** are:

Option	Description
-c	Duplicates in the output files any comments it finds in the original file. If this option is not specified, comment lines are omitted.
-f ***filename***	Specifies the name (and path, if necessary) of AG configuration file to translate. Default is *ag.cfg*. If no path is specified, *ag2oam* searches first in the current directory and then in the paths specified with the AGLOAD environment variable.
-p[***m***[..***n***]=] ***product***	Specifies the AG product type for board(s) ***m***...***n***. This option can appear on the command line as many times as necessary. If you do not specify board numbers, the specified product types are used for all boards. Displaying board product types describes how to get a list of valid values for ***product***.
-?	Displays the Help screen and terminates.
-h	Displays the Help screen and terminates.

For example, with the configuration listed in step 1, enter:

```
ag2oam -f myfile.cfg -p0..1=AG_2000_T1 -p2=AG_4000C_T1
```

If the operation is successful, *ag2oam* returns without a message. *ag2oam* outputs the following files, in the same path as the source file:

- A system configuration file, listing all boards from the AG configuration file. This file is named ***oldname**_oamsys.cfg*, where ***oldname*** is the name of the AG configuration file, minus the extension. For example: *myfile_oamsys.cfg*
- One or more keyword files, one for each board listed in the AG configuration file. This file is named ***oldname**_Board_**n**.cfg*, where ***oldname*** is the name of the AG configuration file, minus the extension, and ***n*** is the number of the board as it appeared in the AG configuration file. For example: *myfile_Board_0.cfg*

  The keyword file for each board is appropriately referenced in the system configuration file, in the section describing the board.

  Note: *ag2oam* assumes that the input AG configuration file is valid. If errors exist in the input file, in most cases they will be propagated in the output files.

Board identification changes

Previously, the board number was the only way of identifying a board in software. This number was assigned in the AG configuration file. With the NMS OAM service, boards are also identified by board names. The board name for a board is assigned when the record for the board is first created in the NMS OAM database. You can specify the name of a board in the system configuration file you supply to *oamsys.* Refer to *Creating a system configuration file* on page 43.

Names are also used for other types of components, such as extended management components (EMCs), board plug-ins, and the NMS OAM Supervisor itself. For details, see *Configuring non-board objects* on page 44.

Most NMS API software still requires board numbers. Within NMS OAM, boards are still assigned unique board numbers, and you can still use this method to identify them in software. Within the NMS OAM service, you can also identify a board using its location (bus and slot), as well as with other information. For details, see *Board identification methods* on page 14.

Hot Swap changes

Previously, Hot Swap was implemented as a Natural Access service (the HSI service). This interface is now implemented as an NMS OAM extended management component (EMC). The following list details changes to the API made as a result of the NMS OAM implementation. For further information, refer to the *NMS OAM Service Developer's Reference Manual*.

- The HSI service is deprecated and is not compatible with NMS OAM.
- The information formerly returned by HSI functions **hsiGetBoardInfo** and **hsiGetLogicalBoardInfo** is now available using other means, as follows:

Information	New source
Board information	**oamBoardGet*XXX*** and **oamBoardLookupBy*XXX*** functions
Hot Swap state	Hot Swap EMC Board.***name***.State keyword

- Hot Swap events are now passed to applications using the same event handling mechanism used for NMS OAM events. Hot Swap events and errors have not changed, except for their prefixes: Hot Swap events have the prefix HSWEVN_ and Hot Swap error codes have the prefix HSWERR_. They are specified in *hswdef.h*.
- A new state was added to the state machine: Unsupported. If a board does not support Hot Swap, it is permanently in this state.
- Hot Swap state names have changed, to be closer to their SNMP equivalents:

Old state name	New state name
NOT PRESENT	Extracted
OFFLINE	OffLine
PREPARATION	OnLinePending
PREPARATION FAILED	Failed
RUNNING	OnLine
DOWNING	OffLinePending
(none)	Unsupported

- The *Hot Swap Developer's Manual* is now obsolete. For Hot Swap runtime information, refer to the Hot Swap topics in the manual you are currently reading. For Hot Swap developer information, refer to the *NMS OAM Service Developer's Reference Manual*.

Tracing changes

Previously, AG driver trace messages were displayed by *agmon* or retrieved using the AGM library API. The debug mask was set using *agmon*'s -x command line option, or using *agtrace* or the AGM library. The CG driver was not supported.

With NMS OAM, AG and CG driver trace messages are displayed by the *oammon* monitoring utility or retrieved through the Natural Access queue. The debug mask is set using *agtrace*.

Trace messages are also logged to *agpierror.log*. Under Windows, this file can be found in *\nms\oam\log*. Under UNIX it is stored in */opt/nms/oam/log*. This file is formatted in the same way as the *agerror.log* file generated with *agmon*. Use *dectrace* to decode ISDN information from this file, as follows:

```
dectrace -f \nms\oam\log\agpierror.log > mytrace.txt
```

A large amount of data is generated in some cases when tracing is enabled. To avoid overloading the system, shut down *oammon* before setting the tracing flags with *agtrace*. In this case, you can get the tracing information from *agpierr.log* as previously described.

Previously, Natural Access dispatcher traces were displayed by *ctdaemon*. *ctdaemon* was also used to set the debug mask. With NMS OAM, dispatcher traces are displayed by *oammon* or retrieved through the Natural Access queue. The debug mask is set using the board keyword DebugMask.

For more information, see the board documentation.

Index